Table of Contents

Table of Contents

Table of Contents

Computing at NAS

Computing Overview

Once you have gone through the steps of getting an allocation and account, setting up your environment on your local machine to securely connect to a NAS high-end computing system; and customizing your NAS environment, then you are ready to utilize our supercomputing resources for actual work.

Reading through the following articles will help you through the next steps:

- Hardware Overviews: Pleiades and Columbia
- Software: Overview
- Porting and Developing: Overview
- anything you need to know about running jobs

Computing Hardware

Pleiades

Pleiades: Introduction

Pleiades is the primary supercomputer at NAS. Originally installed in 2008 with 51,200 cores, it has been further expanded at various stages. The following articles provide hardware information at varying levels of detail:

- Pleiades Hardware Overview - a high-level overview of the Pleiades system architecture, including resource summaries of the compute and front-end nodes, the interconnect, and the storage capacity.
- Pleiades Configuration Details - focuses on the hardware hierarchy (from the processors to the whole cluster) and provides more detailed configuration statistics on the processors and their associated memory.
- Harpertown Processors, Nehalem-EP Processors, and Westmere Processors (3 articles) - provide configuration diagrams and additional information such as core labeling, instruction set, hyperthreading, and Turbo Boost, for each of Pleiades' three processor types.
- Comparison among Harpertown, Nehalem-EP, and Westmere - points out the differences and similarities among the three processor types.
- Pleiades Home Filesystem - information on quota and backup policies on the home filesystem.
- Pleiades Lustre Filesystems - details the configurations of the Lustre filesystems and users' quotas on these filesystems.
- Pleiades Interconnect - information on the topology, latency, and bandwidth of the Pleiades InfiniBand fabric.
- Pleiades Front-End Usage Guidelines - guidelines on using the front-end nodes and bridge nodes.

Pleiades Hardware Overview

Pleiades, the seventh most powerful supercomputer in the world, represents NASA's state-of-the-art technology for meeting the agency's supercomputing requirements, enabling NASA scientists and engineers to conduct modeling and simulation for NASA missions. This distributed-memory SGI ICE cluster is connected with InfiniBand in a dual-plan hypercube technology.

This system contains the following types of Intel Xeon processors: X5670 (Westmere), X5570 (Nehalem), and E5472 (Harpertown). Pleiades is named after the astronomical open star cluster of the same name.

System Architecture

- Manufacturer - SGI
- 184 racks (11,766 nodes)
- 1.33 Pflop/s peak cluster
- 1.09 Pflop/s sustained performance (June 2011)
- Total cores: 112,540
- Total memory: 188TB
- Nodes
    - 4,544 Westmere nodes
        - 2 six-core processors per node
        - Xeon X5670 (Westmere) processors
        - Processor speed - 2.93GHz
        - Cache - 12MB Intel Smart Cache for 6 cores
        - Memory Type - DDR3 FB-DIMMs
        - 2GB per core, 24GB per node
        - InfiniBand® QDR host channel adapter
    - 1,280 Nehalem nodes
        - 2 quad-core processors per node
        - Xeon X5570 (Nehalem) processors
        - Processor speed - 2.93GHz
        - Cache - 8MB Intel Smart Cache for 4 cores
        - Memory Type - DDR3 FB-DIMMs
        - 3GB per core, 24GB per node
        - InfiniBand® DDR host channel adapter
    - 5,888 Harpertown nodes
        - 2 quad-core processors per node
        - Xeon E5472 (Harpertown) processors
        - Processor speed - 3GHz
        - Cache - 6MB per pair of cores
        - Memory Type - DDR2 FB-DIMMs
        - 1GB per core, 8GB per node
        - InfiniBand® DDR host channel adapter

Subsystems

- 14 front-end nodes
    - ◊ 2 quad-core processors per node
    - ◊ Xeon E5472 (Harpertown) processors
    - ◊ Processor speed - 3GHz
    - ◊ 16 GB per node
    - ◊ 1 Gigabit Ethernet connection
- 2 bridge nodes
    - ◊ 2 quad-core processors per node
    - ◊ Xeon E5472 (Harpertown) processors
    - ◊ Processor speed - 3GHz
    - ◊ 64 GB per node
    - ◊ 10 Gigabit Ethernet connection
- 1 PBS server
    - ◊ 2 quad-core processors per node
    - ◊ Xeon E5472 (Harpertown) processors
    - ◊ Processor speed - 3GHz
    - ◊ 16 GB per node

Interconnects

- Internode - InfiniBand, with all nodes connected in a partial 11D hypercube
- Two independent InfiniBand fabrics (ib0, ib1)
- Infiniband DDR, QDR
- Gigabit Ethernet management network

Storage

- SGI® InfiniteStorage NEXIS 9000 home filesystem
- 12 DDN 9900 RAIDs - 6.9 PB total
- 7 Oracle Lustre cluster-wide filesystems, each containing:
    - ♦ 1 Metadata server (MDS)
    - ♦ 8 Object Storage Servers (OSS)
    - ♦ 60 - 120 Object Storage Targets (OST)

Operating Environment

- Operating system - SUSE® Linux®
- Job Scheduler - PBS®
- Compilers - Intel and GNU C, C++ and Fortran
- MPI - SGI MPT, MVAPICH2, Intel MPI

Related Links

Links related to the Pleiades system.

- Pleiades Configuration Details
- Pleiades Front-End Usage Guidelines

Pleiades Configuration Details

DRAFT

This article is being reviewed for completeness and technical accuracy.

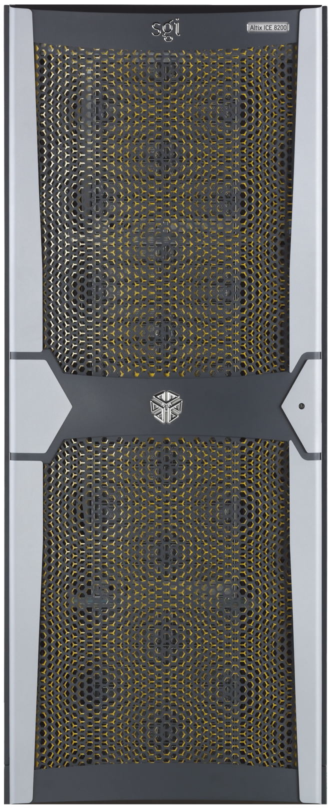
Pleiades Hardware Hierarchy

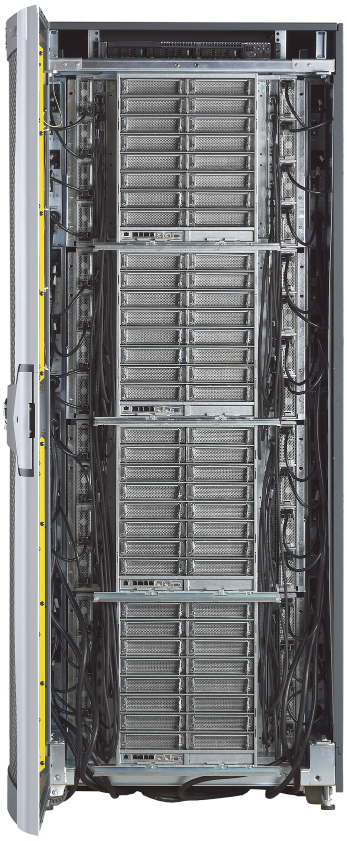
The hardware hierarchy from a single processor to the whole Pleiades supercluster is described below:

- one quad-core (Harpertown, Nehalem-EP) or 6-core (Westmere) processor per socket
- 2 sockets in 1 node
- 16 compute nodes (labeled as n0 - n15) in 1 IRU (individual rack units)
- 4 IRUs (labeled as i0 - i3) in 1 rack
- 91 Harpertown racks (labeled as r1-r91);
  20 Nehalem-EP racks (labeled as r161-170, r177-186):
  and 73 Westmere racks (labeled as r129-160, r171-176, r187-218, r219, r221-r222)
  in the Pleiades supercluster

The nomenclature of a Pleiades compute node is based on which rack and IRU it is on. For example, r1i0n15 is the node 15 in IRU 0 of rack 1.

Below are the front views of a typical rack:


Front view of a rack


Front-open view of a rack (includes 4 IRUs)

Processor and Memory Subsystems Statistics

Below are detailed configuration statistics for the processor and memory subsystems for all Pleiades nodes:

Pleiades Processor and Memory Subsustems Statistics

Hostname	**pfe[1-12] bridge[1-2]**	**pbspl1, pbspl3**	**r[1-91]i[0-3]n[0-15]**	**r[161-170,177-186] i[0-3]n[0-15]**	**r[129-160,1 219, 221-22 i[0-3]n[0-15**
Function	front-end * bridge node with Columbia CXFS filesystems mounted	PBS server	compute	compute	compute
Architecture	ICE 8200EX	ICE 8200EX	ICE 8200EX	ICE 8200EX	ICE 8400EX
			Processor		
CPU	**Quad-Core Xeon E5472 (Harpertown)**	**Quad-Core Xeon E5472 (Harpertown)**	**Quad-Core Xeon E5472 (Harpertown)**	**Quad-Core Xeon X5570 (Nehalem-EP)**	**6-Core Xeon X567 (r221-222)**

					(Westmere
CPU-Clock	**3.00 GHz**	**3.00 GHz**	**3.00 GHz**	**2.93 GHz**	**2.93/3.07 (r**
Floating Point Operations per cycle per Core	4	4	4	4	4
# of Cores/blade (or node)	8	8	8	8	12
Total # of nodes	.	.	5,824	1,280	4,672
Total # of Cores	.	.	46,592	10,240	56,064
			Memory		
L1 Cache	Local to each core; Instruction cache: 32K Data cache: 32K; 32B/cycle;	Local to each core; Instruction cache: 32K Data cache: 32K; 32B/cycle;	Local to each core; Instruction cache: 32K Data cache: 32K; 32B/cycle;	Local to each core; Instruction cache: 32K Data cache: 32K; 32B/cycle;	Local to eac Instruction c Data cache 32B/cycle;
L2 Cache	12MB on-die for the Quad-Core; 6MB per core pair; shared by the two cores.	12MB on-die for the Quad-Core; 6MB per core pair; shared by the two cores. L2 Cache speed: 3 GHz	12MB on-die for the Quad-Core; 6MB per core pair; shared by the two cores. L2 Cache speed: 3 GHz	256 KB per core	256 KB per
L3 Cache	N/A	N/A	N/A	8 MB shared by the four cores	12 MB shar cores
TLB	local to each core	local to each core	local to each core	local to each core	local to eac
Default Page Size	4 KB	4 KB	4 KB	4 KB	4 KB
Local Memory/Core	**2 GB (pfe[1-12]); 8 GB (bridge[1-2]; Fully Buffered DDR2 DIMM**	**2 GB**	**1 GB**	**3 GB; DDR3**	**2 GB: DDR**
		16 GB	8 GB	24 GB	24 / 48(r21

Total Memory/node	16 GB (pfe[1-12]); 64 GB (bridge[1-2])				
Front-Side Bus	1600 MHz; 25.6 GB/sec read 12.8 GB/sec write	1600 MHz; 25.6 GB/sec read 12.8 GB/sec write	1600 MHz; 25.6 GB/sec read 12.8 GB/sec write	N/A	N/A
Memory Controller	N/A	N/A	N/A	32 GB/sec read/write	32 GB/sec r
QuickPath Interconnect	N/A	N/A	N/A	25.6 GB/sec	25.6 GB/se

One of the Harpertown racks, rack 32, provides 16 GB of memory per node, double the size of per-node memory available in the other Harpertown racks.

Related articles:

Pleiades Hardware Overview

Harpertown Processors

Nehalem-EP Processors

Westmere Processors

Comparison among Harpertown, Nehalem-EP, and Westmere

Pleiades Home Filesystem

Pleiades Lustre Filesystems

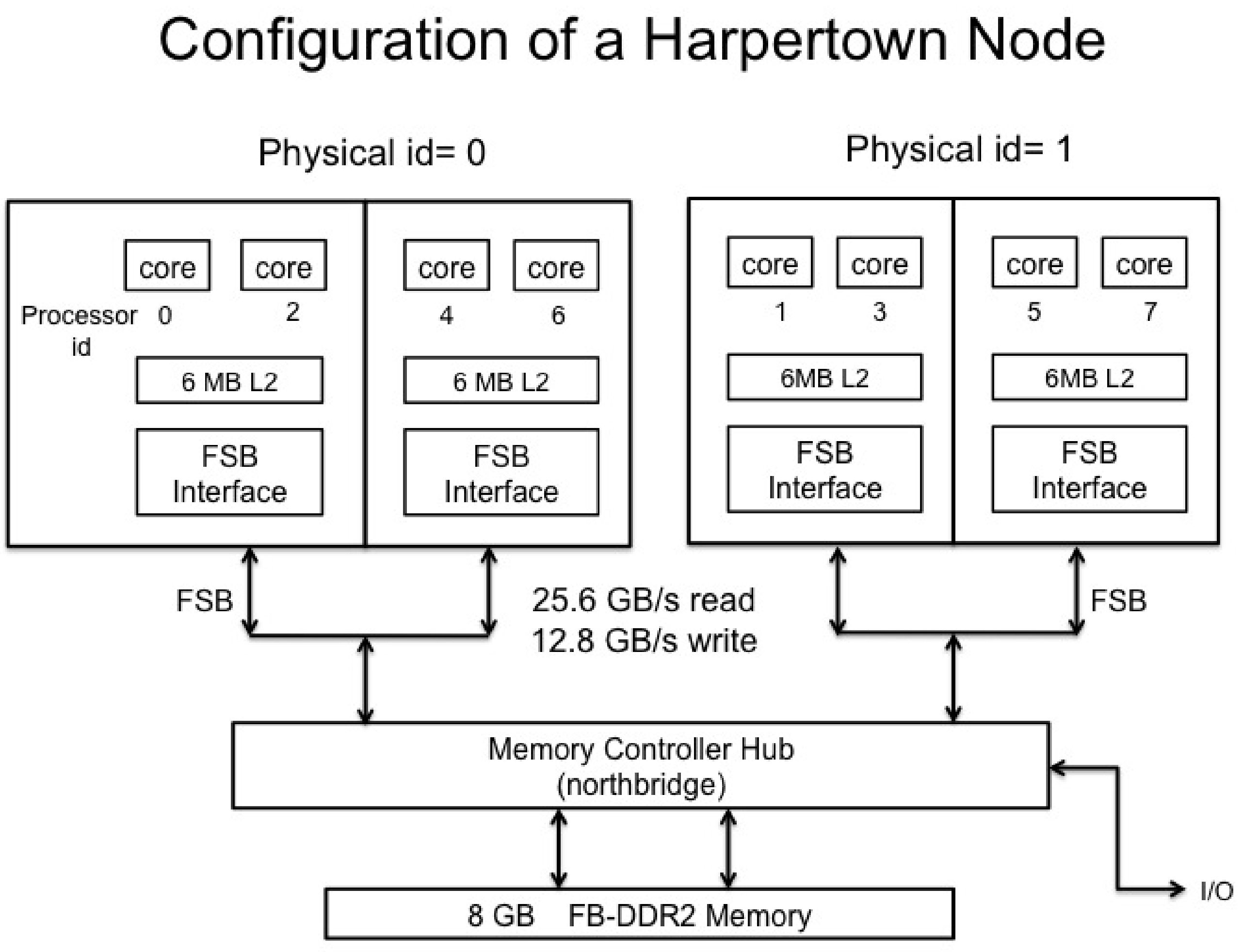
Pleiades Front-End Usage Guidelines

Harpertown Processors

DRAFT

This article is being reviewed for completeness and technical accuracy.

Configuration of a Harpertown node:



Core Labeling:

The core labeling as shown in this diagram is obtained from the command *cat /proc/cpuinfo*. Note that in the first socket (i.e., phyiscal id=0), the four cores are labeled 0, 2, 4, and 6, and are not contiguous. Similarly, in the second socket (physical id=1), they are labeled as 1, 3, 5, and 7. In addition, each core pair (0,2), (4,6), (1,3) and (5,7) shares a 6MB L2 cache.

For performance consideration, care must be taken if one tries to use tools such as *dplace* to pin processes to specific processors. Be aware of the non-contiguous nature of the labeling and the sharing of L2 cache per core pair. Also, when using the SGI MPT library, the environment variable **MPI_DSM_DISTRIBUTE** has been set to *off* for the Harpertown nodes since setting MPI_DSM_DISTRIBUTE to *on* causes the processes to be pinned to processors in a contiguous order. For example, MPI ranks 0-7 are pinned to processors 0-7, respectively. This results in bad performances for most applications.

SSE4 Instruction Set:

Intel's Streaming SIMD Extensions 4.1 (SSE4.1) instruction set is included in the Harpertown processors.

Since the instruction set is upward compatible, an application which is compiled with -xSSE4.1 (with Intel version 11 compiler) can run on either Harpertown or Nehalem-EP or Westmere processors. An application which is compiled with -xSSE4.2 can run ONLY on Nehalem-EP or Westmere processors.

If you wish to have a single executable that will run on any of the three Pleiades processor types with suitable optimization to be determined at run time, you can compile your application with -O3 -ipo -axSSE4.2,xSSE4.1

Hyperthreading:

Not available.

Turbo Boost:

Not available.

Front-Side Bus

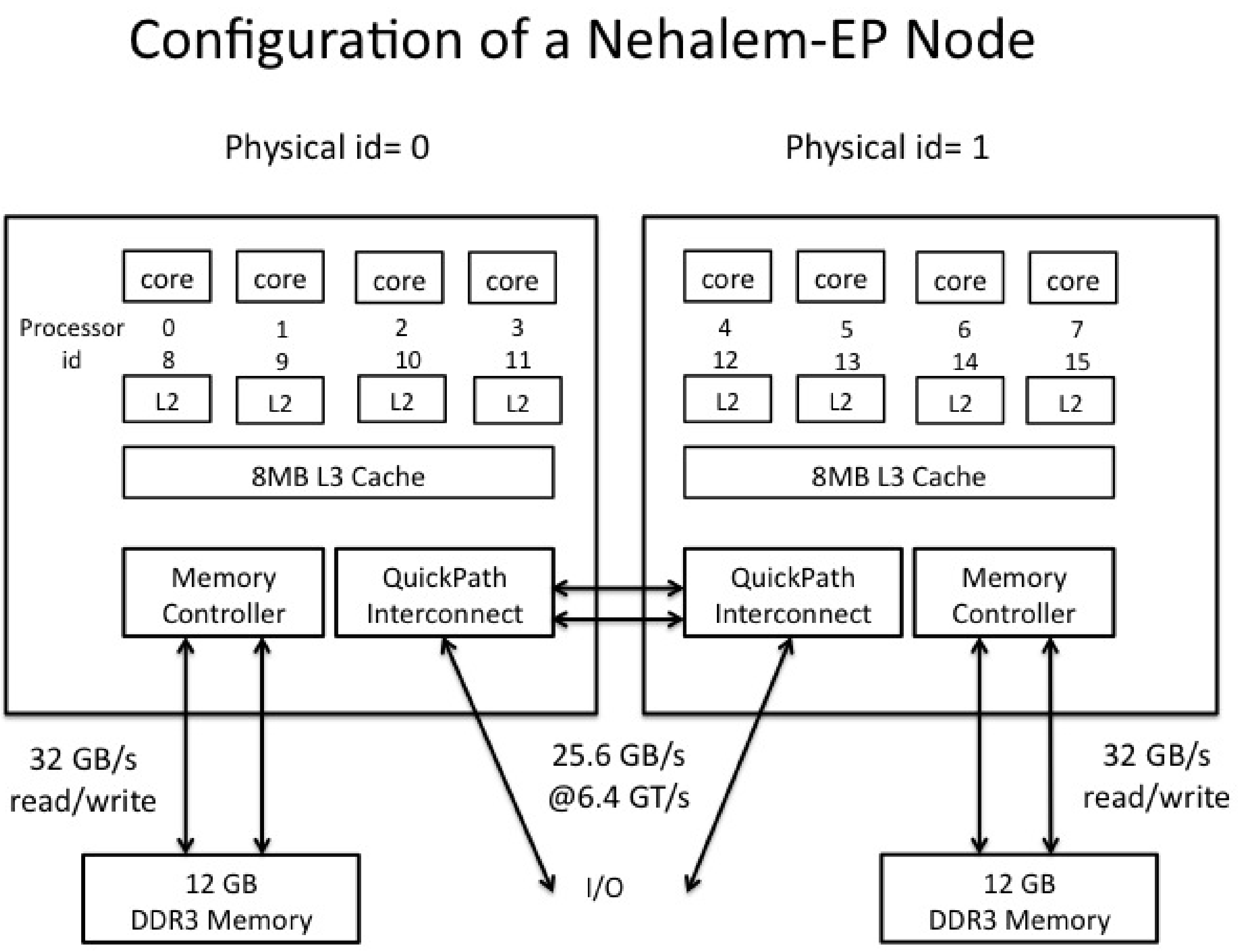
The Harpertown (Quad-Core Intel Xeon Processor E5472) processors at NAS use 1600 MHz Front-Side Bus (FSB). The processor transfers data four times per bus clock (4X data transfer rate, as in AGP 4X). Along with the 4X data bus, the address bus can deliver addresses two times per bus clock and is referred to as a double-clocked or a 2X address bus. In addition, the Request Phase completes in one clock cycle. Working together, the 4X data bus and 2X address bus provide a data bus bandwidth of up to 12.8 GBytes per second. The FSB is also used to deliver interrupts.

Nehalem-EP Processors

DRAFT

This article is being reviewed for completeness and technical accuracy.

Configuration of a Nehalem-EP node:



Core Labeling:

Unlike Harpertown, the core labeling in Nehalem-EP (and also Westmere) is contiguous. That is, cores 0-3 are in first socket and cores 4-7 are in the second socket.

When using the SGI MPT library, the enviroment variable **MPI_DSM_DISTRIBUTE** is set to *on* by default for the Nehalem-EP (and also Westmere) nodes.

SSE4 Instruction Set:

Intel's Streaming SIMD Extensions 4.2 (SSE4.2) instruction set is included in the Nehalem-EP processors.

Since the instruction set is upward compatible, an application that is compiled with -xSSE4.1 (with Intel version 11 compiler) can run on either Harpertown or Nehalem-EP or Westmere processors. An application that is compiled with -xSSE4.2 can run ONLY on Nehalem-EP or Westmere processors.

If you wish to have a single executable that will run on any of the three Pleiades processor types with suitable optimization to be determined at run time, you can compile your application with -O3 -ipo -axSSE4.2,xSSE4.1

Hyperthreading:

On Nehalem-EP (and also Westmere), hyperthreading is available by user request, for example by asking for more than 8 MPI ranks per Nehalem-EP node.

When hyperthreading is requested, the OS views each physical core as two logical processors and can assign two threads to it.

Preliminary benchmarking by NAS shows that many jobs would benefit from using hyperthreading. Therefore, it is currently turned ON, meaning that it is available if a job requests it.

Mapping of Physical Cores and Logical Processor IDs

Physical id	**Core id**	**Processor id** Hyperthreading **OFF**	**Processor id** Hyperthreading **ON**
0	0	0	0 ; 8
0	1	1	1 ; 9
0	2	2	2 ; 10
0	3	3	3 ; 11
1	4	4	4 ; 12
1	5	5	5 ; 13
1	6	6	6 ; 14
1	7	7	7 ; 15

With hyperthreading, one can run an MPI code with 16 processes instead of just 8 per Nehalem-EP node. Each of the 16 processes will be assigned to run on one logical processor. In reality, two processes are running on the same physical core. If one process does not keep the functional units in the core busy all the time and can share the resources in the core with another process, then running in this mode will take less than 2 times the walltime compared to running only 1 process on the core. This can improve the overall throughput as demonstrated in the following example:

Example: Consider the following scenario with a job that uses 16 MPI ranks. Without hyperthreading we would use:

```
#PBS -lselect=2:ncpus=8:mpiprocs=8 -lplace=scatter:excl
```

and the job will use 2 nodes with 8 processes per node. Suppose that the job takes 1000 seconds when run this way. If we run the job with hyperthreading, e.g.:

```
#PBS -lselect=1:ncpus=16:mpiprocs=16 -lplace=scatter:excl
```

then the job will use 1 node with all 16 processes running on that node. Suppose this job takes 1800 seconds to complete.

Without hyperthreading, we used 2 nodes for 1000 seconds (a total of 2000 node-seconds); with hyperthreading we used 1 node for 1800 seconds (1800 node-seconds). Thus, under these circumstances, if you were interested in getting the best wall-clock time performance for a single job, you would use two nodes without hyperthreading. However, if you were interested in minimizing resource usage, especially with multiple jobs running simultaneously, use of hyperthreading would save you 10%.

An added benefit of using fewer nodes with hyperthreading, is that when Pleiades is loaded with many jobs, asking for half as many nodes may allow your job to start running sooner, resulting with an improvement in the throughput of your jobs.

Caution: Hyperthreading does not benefit all applications. Some applications may also show improvement with some process counts but not with other process counts (e.g., a 256-process Overflow job shows benefit with hyperthreading, while a 32-process Overflow job does not). There may also be other unforeseen issues with hyperthreading. Users should test their applications with and without hyperthreading before making a choice for production runs. If your application runs more than 2 times slower with hyperthreading than without hyperthreading, then it should not be used.

Turbo Boost:

On Nehalem-EP (and also Westmere), Turbo Boost is available.

When Turbo Boost is enabled, idle cores are turned off and and power is channeled to the cores that are active, making them more efficient. The net effect is that the active cores perform above their clock speed (i.e., overclocked).

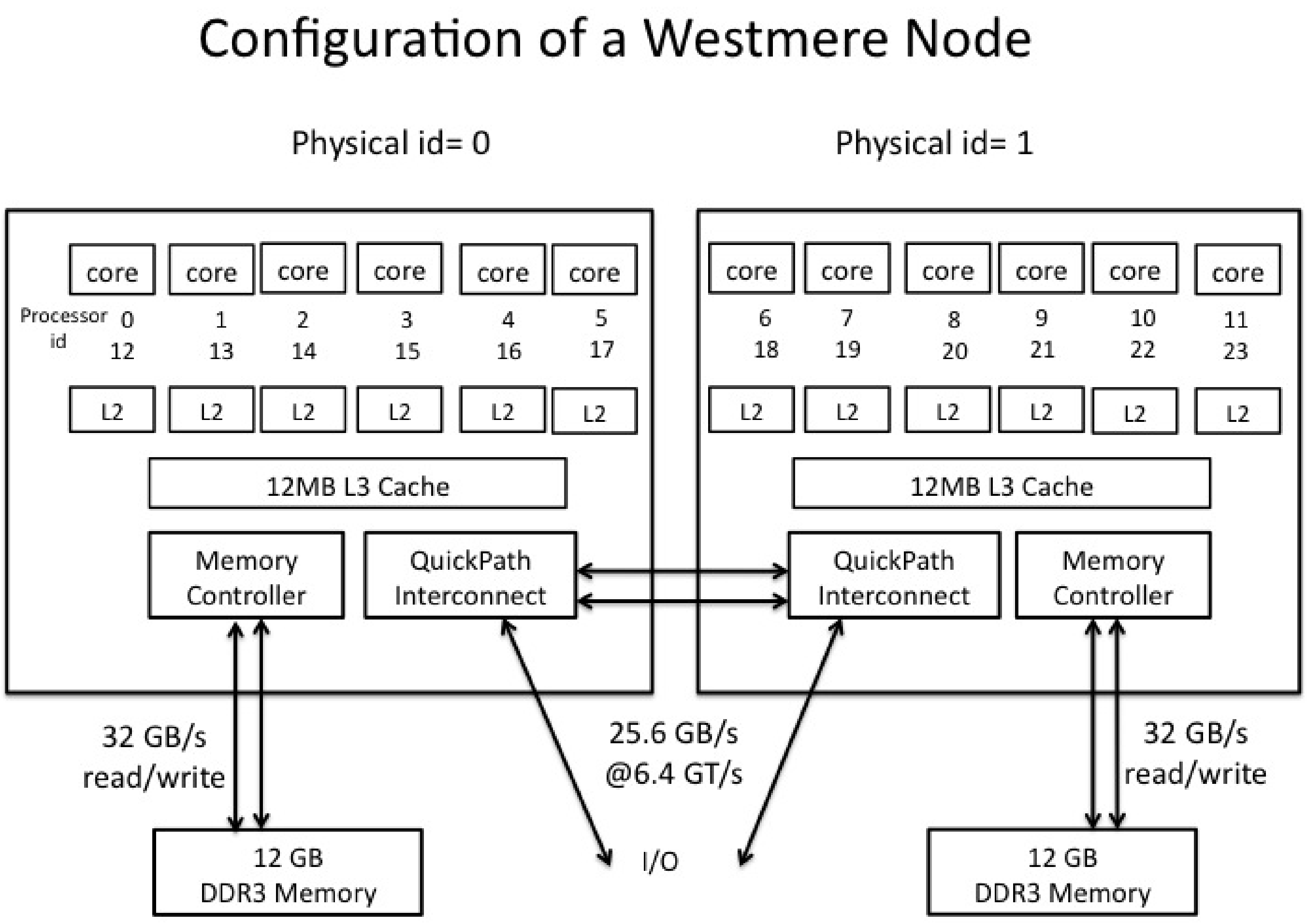
Turbo Boost mode is set up in the system BIOS. It is currently set to OFF.

Westmere Processors

DRAFT

This article is being reviewed for completeness and technical accuracy.

Configuration of a Westmere node:



Core Labeling:

Unlike Harpertown, the core labeling in Westmere is contiguous. That is, cores 0-5 are in first socket and cores 6-11 are in the second socket.

When using the SGI MPT library, the enviroment variable **MPI_DSM_DISTRIBUTE** is set to *on* by default for the Westmere nodes.

SSE4 Instruction Set:

Intel's Streaming SIMD Extensions 4.2 (SSE4.2) instruction set is included in the Westmere processors.

Since the instruction set is upward compatible, an application that is compiled with -xSSE4.1 (with Intel version 11 compiler) can run on either Harpertown or Nehalem-EP or Westmere processors. An application that is compiled with -xSSE4.2 can run ONLY on Nehalem-EP or Westmere processors.

If you wish to have a single executable that will run on any of the three Pleiades processor types with suitable optimization to be determined at run time, you can compile your application with -O3 -ipo -axSSE4.2,xSSE4.1

Hyperthreading:

Hyperthreading is available by user request on the Westmere nodes (for example, by asking for more than 12 ranks per node).

Turbo Boost:

Turbo Boost is set to *off* on the Westmere nodes.

Comparison among Harpertown, Nehalem-EP and Westmere

DRAFT

This article is being reviewed for completeness and technical accuracy.

Among the three processor types used in Pleiades, Nehalem-EP and Westmere are very similar to each other, while Harpertown is significantly different from the other two.

The main differences between Nehalem-EP and Westmere processors are:

- Both Nehalem-EP and Westmere have 24 GB of memory per node. However, there are 8 cores per Nehalem-EP node vs 12 cores per Westmere node, resulting in more memory per core for Nehalem-EP (3 GB/core) than Westmere (2 GB/core).

- The size of the L3 cache is 8 MB per quad-core for Nehalem-EP while it is 12 MB per 6-core for Westmere.

- For inter-node communication, two devices are involved: the Infiniband switches and the host channel adapter chip (HCA). For both the Nehalem-EP and Westmere racks, there are two SGI Infiniband QDR switches per half-IRU (which comprises 8 nodes). One of the switches is used for ib0 (used mainly for MPI communication), and the other for ib1 (used mainly for IO). The maximum raw data transfer rate through these switches is 40 Gigabits per second (Gbps). However, the HCA on each motherboard (one node per motherboard) is different for Nehalem-EP and Westmere. For Nehalem-EP, a 4x DDR HCA with a raw data transfer rate of 20 Gbps is used. For Westmere, a 4x QDR HCA with a rate of 40 Gbps is used. This difference results in better inter-node communication performance between the Westmere nodes than between the Nehalem-EP nodes.

  Note: The communication path between pairs of nodes vary, depending on where the nodes are relative to each other. For example:

  - ♦ two nodes on the same half-IRU: HCA to IB switch on the half-IRU to HCA.
  - ♦ two nodes on the same IRU but different half-IRUs: HCA to IB switch (of one half-IRU) to IB switch (of the other half-IRU) to HCA.

The main differences between Harpertown and Nehalem-EP/Westmere processors are:

- The processor labeling in Harpertown is not contiguous. On the contrary, the labeling in Nehalem-EP/Westmere is contiguous.

- Nehalem-EP/Westmere incorporates the SSE 4.2 SIMD instructions, which adds 7 new instructions to the SSE 4.1 set in Harpertown.

- Every two cores in Harpertown share a common L2 cache, while every core in Nehalem-EP/Westmere has its own private L2 cache. In addition, there is a L3 cache shared by the four cores in each socket of Nehalem-EP (or by the 6 cores in each socket of Westmere), while there is none for Harpertown.

- The Nehalem-EP based nodes have 3 GB/core (i.e., 24 GB/node) of memory as compared to 1 GB/core (i.e., 8 GB/node) in most of the Harpertown-based nodes in Pleiades.

  The Westmere based nodes have 2 GB/core (i.e., 24 GB/node) of memory as compared to 1 GB/core (i.e., 8 GB/node) in most of the Harpertown-based nodes in Pleiades.

- Nehalem-EP/Westmere, with a higher ratio of memory bandwidth to processor speed, is a better balanced system than the Harpertown.

  The key features which enable this improvement are the Intel QuickPath Interconnect, which provides communication with the other processor on the same node, and an integrated memory controller. Together they result in a higher aggregate bandwidth.

  In addition, each Nehalem-EP/Westmere core has its own L1 and L2 cache which helps to decrease the number of stalls in a data path. The data pre-fetch algorithm for L2 and L3 caches has been substantially reworked to achieve more effective data loads.

- Hyperthreading and TurboBoost are additional features on Nehalem-EP/Westmere, but not for Harpertown. Hyperthreading is available by user request for Nehalem-EP/Westmere. Turbo Boost is set to *off* for Nehalem-EP/Westmere.

Pleiades Home Filesystem

DRAFT

This article is being reviewed for completeness and technical accuracy.

The home file system on Pleiades (/u/username) is an SGI NEXIS 9000 filesystem. It is NFS-mounted on all of the Pleiades front-ends, bridge nodes and compute nodes.

Once a user is granted an account on Pleiades, the home directory is set up automatically during his/her first login.

Quota and Policy

Disk space quota limits are enforced on the home filesystem. By default, the soft limit is 8GB and the hard limit is 10GB. There are no inode limits on the home filesystem.

To check your quota and usage on your home filesystem, do:

```
%quota -v
Disk quotas for user username (uid xxxx):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
saturn-ib1-0:/mnt/home2
                7380152  8000000 40000000          190950       0       0
```

The quota policy for NAS states that if you exceed the soft quota, an email will be sent to inform you of your current usage and how much of your grace period remains. It is expected that a user will occasionally exceed their soft limit as needed, however after 14 days, users who are still over their soft limit will have their batch queue access to Pleiades disabled. If you believe that you have a long-term need for higher quota limits, you should send an email justification to support@nas.nasa.gov. This will be reviewed by the HECC Deputy Project Manager, Bill Thigpen, for approval.

The quota policy for NAS can be found here.

Backup Policy

Files on the home filesystem are backed up daily.

Pleiades Lustre Filesystems

Pleiades has several Lustre filesystems (/nobackupp[10-60]) that provide a total of about 3 PB of storage and serve thousands of cores. These filesystems are managed under Lustre software version 1.8.2.

Lustre filesystem configurations are summarized at the end of this article.

Which /nobackup should I use?

Once you are granted an account on Pleiades, you will be assigned to use one of the Lustre filesystems. You can find out which Lustre filesystem you have been assigned to by doing the following:

```
pfe1% ls -l /nobackup/your_username
lrwxrwxrwx 1 root root 19 Feb 23  2010 /nobackup/username -> /nobackupp30/username
```

In the above example, the user is assigned to /nobackupp30 and a symlink is created to point the user's default /nobackup to /nobackupp30.

TIP: Each Pleiades Lustre filesystem is shared among many users. To get good I/O performance for your applications and avoid impeding I/O operations of other users, read the articles: Lustre Basics and Lustre Best Practices.

Default Quota and Policy on /nobackup

Disk space and inodes quotas are enforced on the /nobackup filesystems. The default soft and hard limits for inodes are 75,000 and 100,000, respectively. Those for the disk space are 200GB and 400GB, respectively. To check your disk space and inodes usage and quota on your /nobackup, use the *lfs* command and type the following:

```
%lfs quota -u username /nobackup/username
Disk quotas for user username (uid xxxx):
   Filesystem  kbytes       quota   limit   grace   files   quota   limit   grace
/nobackup/username 1234  210000000 420000000    -      567   75000  100000       -
```

The NAS quota policy states that if you exceed the soft quota, an email will be sent to inform you of your current usage and how much of your grace period remains. It is expected that users will occasionally exceed their soft limit, as needed; however after 14 days, users who are still over their soft limit will have their batch queue access to Pleiades disabled.

If you anticipate having a long-term need for higher quota limits, please send a justification via email to support@nas.nasa.gov. This will be reviewed by the HECC Deputy Project Manager for approval.

For more information, see also, Quota Policy on Disk Space and Files.

NOTE: If you reach the hard limit while your job is running, the job will die prematurely without providing useful messages in the PBS output/error files. A Lustre error with code -122 in the system log file indicates that you are over your quota.

In addition, when a Lustre filesystem is full, jobs writing to it will hang. A Lustre error with code -28 in the system log file indicates that the filesystem is full. The NAS Control Room staff normally will send out emails to the top users of a filesystem asking them to clean up their files.

Important: Backup Policy

As the names suggest, these filesystems are not backed up, so any files that are removed *cannot* be restored. Essential data should be stored on Lou1-3 or onto other more permanent storage.

Configurations

In the table below, /nobackupp[10-60] have been abbreviated as p[10-60].

Pleiades Lustre Configurations

Filesystem	p10	p20	p30	p40	p50	p60
# of MDSes	1	1	1	1	1	1
# of MDTs	1	1	1	1	1	1
size of MDTs	1.1T	1.0T	1.2T	0.6T	0.6T	0.6T
# of usable inodes on MDTs	~235x10^6	~115x10^6	~110x10^6	~57x10^6	~113x10^6	~123x10^6
# of OSSes	8	8	8	8	8	8
# of OSTs	120	60	120	60	60	60
size/OST	7.2T	7.2T	3.5T	3.5T	7.2T	7.2T
Total Space	862T	431T	422T	213T	431T	431T
Default Stripe Size	4M	4M	4M	4M	4M	4M
Default Stripe Count	1	1	1	1	1	1

NOTE: The default stripe count and stripe size were changed on January 13, 2011. For directories created prior to this change, if you did not explictly set the stripe count and/or stripe size, the default values (stripe count 4 and stripe size 1MB) were used. This means that files created prior to January 13, 2011 had those old default values. After this date, directories without an explicit setting of stripe count and/or stripe size adopted the new stripe count of 1 and stripe size of 4MB. However, the old files in that directory will retain their old default values. New files that you create in these directories will adopt the new default values.

Pleiades Front-End Usage Guidelines

DRAFT

This article is being reviewed for completeness and technical accuracy.

The front-end systems pfe[1-12] and bridge[1,2] provide an environment that allows you to get quick turnaround while performing the following:

- file editing
- compiling
- short debugging and testing session
- batch job submission to the compute systems

Bridge[1,2], with 4 times the memory on pfe[1-12] and better interconnects, can also be used for the following two functions:

1. **Post processing**

   These nodes have 64-bit versions of IDL, Matlab, and Tecplot installed and have 64 GB of memory (4 times the amount of memory on pfe[1-12]). The bridge nodes will run these applications much faster than on pfe[1-12].

2. **File transfer between Pleiades and Columbia or Lou**

   Note that both the Pleiades Lustre filesystems (/nobackupp[10-70]) and the Columbia CXFS filesystems (/nobackup1[1-h], /nobackup2[a-i]) are mounted on the bridge nodes.

   To copy files between the Pleiades Lustre and Columbia CXFS filesystems, log in to bridge[1,2] and use the *cp* command to perform the transfer. The 10 Gigabit Ethernet (GigE) connections on the two bridge nodes are faster than the 1 GigE used on pfe[1-12], therefore, file transfer out of Pleiades is improved when using the bridge nodes.

   File transfers from bridge[1,2] to Lou[1,2] will go over the 10 GigE interface by default. The commands *scp*, *bbftp*, and *bbscp* are available to do file transfers. Since *bbscp* uses almost the same syntax as *scp*, but performs faster than *scp*, we recommend using *bbscp* over *scp* in cases where you do not require the data to be encrypted when sent over the network.

   ```
   The pfe systems ([pfe1-12]) have a 1 GigE connection, which
   can be saturated by a single secure copy (scp). You will see
   bad performance whenever more than one file transfer is
   happening. Use of bridge1 and bridge2 for file transfers is
   ```

```
strongly recommended.
```

File transfers from the compute nodes to Lou must go through pfe[1-12] or bridge[1,2] first, although going through bridge[1,2] is preferred for performance consideration. See Transferring Files from the Pleiades Compute Nodes to Lou for more information.

When sending data to Lou[1-2], please keep your largest individual file size under 1 TB, as large files will keep all of the tape drives busy, preventing other file restores and backups. To prevent the filesystems on Lou[1-2] from filling up, please limit total data transfers to 1 TB and then wait an hour before continuing. This allows the tape drives to write the data to tape.

Additional restrictions apply to using these front-end systems:

1. No MPI jobs are allowed to run on pfe[1-12], bridge[1,2]

2. A job on pfe[1-12] should not use more than 8 GB. When it does, a courtesy email is sent to the owner of the job.

3. A job on bridge[1,2] should not use more than 56 GB. When it does, a courtesy email is sent to the owner of the job.

Pleiades Interconnect

DRAFT

This article is being reviewed for completeness and technical accuracy.

Topology

InfiniBand (IB) is used for inter-node communication among all of the Pleiades nodes. A key feature of InfiniBand permits remote direct memory access (RDMA) between processing nodes, allowing direct access to other nodes' memory. This allows developers and application owners to bypass the TCP/IP stack, accelerating the application performance. Two devices are involved in the interconnect: the Mellanox ConnectX host channel adapter chip (HCA) on the motherboard of each node and the Mellanox IB switches. There are two IB switches per half- IRU (which includes 8 nodes). One of the switches is involved in the ib0 fabric, which is used mainly for MPI communication. The other is involved in the ib1 fabric which is used mainly for I/O. InfiniBand uses subnet manager (SM) software to manage the InfiniBand fabric and to monitor interconnect performance and health at the fabric level.

The network topology of each IB fabric of Pleiades is a partial 11-D hypercube. In a 11-D hypercube, each switch has 11 direct connections with 11 other specific switches in the network.

The ib1 hypercube fabric is extended by a set of nine switches connected to the Lustre servers (one for the MDSes, and eight for the OSSes). The plan is for each rack to connect directly to one of the OSS switches and each group of eight racks to connect directly to the MDS switch. Currently, most of the racks are connected this way, but some remain to be connected.

Another set of nine switches on the ib1 fabric provides direct access between hyperwall visualization nodes and Pleiades nodes and Lustre servers.

Latency

The shortest communication path in a Pleiades IB fabric will be for any two nodes located in the same half-IRU of the same rack such that the communication only needs to go through 1 switch. For a fully populated 11-D hypercube, the optimum communication path between any two nodes which are not in the same half-IRU varies from going through 2 to 12 switches, depending on which racks and half-IRUs the two nodes reside. Since the Pleiades IB fabric is not a full 11-D hypercube, some connections are missing that would facilitate the optimum path between some nodes, therefore, it is possible that some communications may go over more than 12 switches.

MPI half Ping-Pong latency starts around 1000 to 1500 ns for communication going through two switches. Each additional switch adds ~100 ns (QDR) to ~150 ns (DDR) to the latency.

Bandwidth

The HCA on each node uses either 4x DDR (double data rate) links or 4x QDR (quad data rate) links. Each link is bi-directional and contains 1 send channel and 1 receive channel. For each direction, the raw data transfer rates for 4x DDR and 4x QDR are 20 Gb/s and 40 Gb/s, respectively. These links use 8b/10b encoding such that every 10 bits sent carry 8 bits of useful data. Thus, for each direction, the effective maximum bandwidth for each node is 16 Gb/s (i.e, 2 GB/s) if 4x DDR HCA is used or 32 Gb/s (i.e. 4 GB/sec) if 4x QDR HCA is used.

The IB switch which every 8 nodes in each half-IRU share through a single path also has similar effective data transfer limits per port: 16 Gb/s for 4x DDR IB switches and 32 Gb/s for 4x QDR IB switches.

The Harpertown and Nehalem-EP nodes use 4x-DDR HCAs while the Westmere nodes use 4x-QDR HCAs. The Harpertown racks use 4x DDR IB switches while the Nehalem-EP and Westmere racks use 4x QDR switches.

These limits also apply to each OSS in the Lustre filesystem. For /nobackupp20 and /nobackupp50, QDR switches are used to connect to the IB fabric and DDR switches are used to connect to the DDNs of the hard disks. For /nobackupp[10,30,40,60], DDR switches are used to connect to both IB fabric and to the DDNs. With DDR switches, the theoretical bandwidth of each OSS is 2 GB/s for each direction. With 8 OSSes per Lustre filesystem, the theoretical peak aggregate bandwidth for each filesystem for each direction would be 16 GB/s. This bandwidth however is reduced to 10 GB/s due to bandwidth that the DDNs can provide. The best benchmark performance obtained for each Pleiades Lustre filesystem is 8 - 10 GB/sec (all read or all write).

The actual I/O bandwidth a user's application experiences is far less than the theoretical peak or even the benchmark data due to factors such as the I/O pattern the application is doing (for example, serial or parallel; for parallel, if the I/O requests are from nodes of different half-IRUs), the number of stripe count used (this affects the maximum aggregate bandwidth provided by the OSSs), how busy the Lustre is handling requests from many users, if there are bad links in the network, etc.

Follow the tips listed in Lustre Best Practices if you are not getting good performances out of the Lustre filesystem.

Columbia

Columbia: Introduction

Columbia, an SGI Altix supercomuter named to honor the crew of Space Shuttle Columbia flight STS-107, has been in production since 2004. In March 2008, the system had 14,136 cores in 24 nodes (Columbia1-Columbia24). When the Pleiades system came into production, the original 20 Columbia nodes (1-20) were retired. Columbia currently comprises 1 front-end node (cfe2) and 4 compute nodes (Columbia21-Columbia24).

The following few articles provide Columbia hardware information at varying levels of detail:

Columbia Hardware Overview provides a high-level overview of the Columbia system architecture, including resource summaries of the compute- and front-end nodes, the interconnect, and storage capacity.

Columbia Configuration Details focuses on more detailed configuration statistics of the processors and their associated memory.

The article Columbia Home Filesystem - provides information on the quota and backup policies on the home filesystem.

The article Columbia CXFS Filesystems - details the configurations of the CXFS filesystems and users' quotas on these filesystems.

In addition, the article Columbia Front-End Usage Guidelines provides guidelines on using the front-end node (cfe2).

Columbia Hardware Overview

DRAFT

This article is being reviewed for completeness and technical accuracy.

Columbia Supercomputer

The Columbia supercluster, which ranked 2nd (51.87 Tflops/s) in the Nov 2004 Top500 list, has been in service for many years. Most of the earlier Columbia nodes (Columbia1 - Columbia20) have been retired. The remaining Columbia nodes (Columbia21-24) continue to serve the NASA community to achieve breakthroughs in science and engineering for the agency's missions and vision for Space Exploration.

Current Columbia System Facts

Manufacturer - SGI

List of nodes for Columbia system

Nodes	Type	Speed	Cache
1 Altix 4700 (512 cores)	Montecito	1.6 GHz	9MB
1 Altix 4700 (2048 cores)	Montecito	1.6 GHz	9MB
2 Altix 4700 (1024 cores)	Montvale	1.6 GHz	9MB
4 Total Compute Nodes (4,608 Total Cores)			

System Architecture

- 40 compute node cabinets
- 30 teraflop/s theoretical peak (original 10,240 system: 63 Tflop/s)

Subsystems

- 1 front-end node

Memory

- Type - double data rate synchronous dynamic random access memory (DDR SDRAM)
- Per Processor (core) - 2GB
- Total Memory - 9TB

Interconnects

- SGI® NUMAlink® interconnected single-system image compute nodes
- Internode
  - ♦ InfiniBand® - 4x (Single Data Rate, Double Data Rate)
  - ♦ 10Gb Ethernet LAN/WAN interconnect
  - ♦ 1Gb Ethernet LAN/WAN interconnect

Storage

- Online - DataDirect Networks® & LSI® RAID, 1PB (raw)
  - ♦ 1 SGI CXFS domains
  - ♦ Local SGI XFS fileystems
- Archival - Attached to high-end computing SGI CXFS SAN filesystem

Operating Environment

- Operating system - SUSE Linux Enterprise
- Job Scheduler - PBS®
- Compilers - C, Intel Fortran, SGI MPT

Columbia Configuration Details

DRAFT

This article is being reviewed for completeness and technical accuracy.

Current Columbia compute nodes, Columbia21-24, are SGI Altix 4700 systems. Detailed information about the processor and memory subsystems of these compute nodes are provided in this article.

Processor and Memory Subsystems Statistics

Below are configuration statistics for the processor and memory subsystems for Columbia21-24:

Columbia Processor and Memory Subsystems Statistics

Hostname	**Columbia21**	**Columbia22**	**Columbia23-24**
Function	compute	compute	compute
Architecture	Altix 4700 (bandwidth configuration)	Altix 4700 (density configuration)	Altix 4700 (density configuration)
	Dual-Core Processor		
Processor	**Itanium2 9040 (Montecito)**	**Itanium2 9040 (Montecito)**	**Itanium2 9150M (Montvale)**
Core-Clock	**1.6 GHz**	**1.6 GHz**	**1.67 GHz**
# of Cores/Node	2	4	4
Nodes/Blade	1	1	1
Total # of Blades	**256**	**512**	**256**
Total # of Cores	**512**	**2048**	**1024**
	Memory		
Local Memory/Node (2 Cores for C21 and 4 Cores for C22, C23-24)	**~3.8 GB**	**~7.6 GB**	**~7.6 GB**
Total Memory	~ 1000 GB	~ 4000 GB	~ 2000 GB
L1 Cache Size/Core	32KB (split into instruction and data cache)	32KB (split into instruction and data cache)	32KB (split into instruction and data cache)
L1 Cache Associativity	4-way	4-way	4-way
L1 Cache Line Size	64 bytes	64 bytes	64 bytes
L2 Cache Size/Core	1MB: instructions 256KB: data	1MB: instructions 256KB: data	1MB: instructions 256KB: data

L2 Cache Associativity	8-way	8-way	8-way
L2 Cache Line Size	128 bytes	128 bytes	128 bytes
L3 Cache Size/Core	9MB	9MB	9MB
L3 Cache Associativity	9-way	9-way	9-way
Default Page Size	16 KB	64 KB	16 KB

Itanium-64 Processors Facts

- The Itanium chip is based on the IA-64 (Intel Architecture, 64 bit) architecture that implements the EPIC (Explicit Parallel Instruction set Computing) technology. With EPIC, an Itanium processor family compiler turns sequential code into parallelized 128-bit bundles that can be directly or explicitly processed by the CPU without having to interpret it further. This explicit expression of parallelism allows the processor to concentrate on executing parallel code as fast as possible, without further optimizations or interpretations. On the contrary, a regular (non-Itanium's processor family) compiler takes a sequential code and examines and optimizes it for parallelism, but then has to regenerate sequential code in a such a way that the processor can re-extract the parallelization from it. The processor then has to read this implied parallelism from the machine code, re-build it, and run it. The parallelism is there, but it is not as obvious to the processor, and more work has to be done by the hardware before it can be utilized.
- Unlike the RISC processors (as used in the SGI Origins) that dedicate an enormous amount of chip real estate and logic to hide cache misses (by allowing instructions to be executed out of order, which works well when the ratio of CPU frequency to memory frequency is relatively small), the EPIC processors rely on the software to make sure that the data is in the proper cache at the proper time. Instructions are issued in order, so there is no hardware mechanism to hide a cache miss.
- The Itanium processors use long instruction words. Specifically, three instructions are grouped into a 128-bit bundle. Each instruction is 41 bits wide. The least significant 5 bits encode a bundle template. The template field encodes (1) the execution units (integer units I, memory units M, floating point units F, and branch units B) needed by the three instructions, and (2) which instructions can be executed in parallel. For the Itanium2 chips, two bundles can be executed per cycle.
- Four memory-load operations per cycle can be delivered from the L2 cache to the floating-point register file. This will completely support two floating-point operations per cycle; this translates into **4 FLOPS per cycle** using the FMA operation.
- Branch predication: without predication, parallelism would be impossible. Instead of waiting for each section of a complex calculation to finish, it is faster if the processor can predict the outcome and proceed on the basis of that prediction. These prediction points are called branches, and current processors try to guess which branch to take. If it predicts correctly, the whole calculation is validated. If it predicts incorrectly, the string has to be thrown out and the calculation starts over. The Itanium processor family architecture minimizes wasted calculations by taking both possible paths to the next branch, where it follows both branches again. When it comes to the correct result it drops the other branch path that it doesn't need, keeps

the branch that it does and it continues on with the calculation.

- Speculative loads; a processor needs to access the memory to get code to execute, but while it fetches this code it is not executing instructions. A processor based on the Itanium processor family architecture specification can look ahead at its instruction and load the required data from the memory early; so, when those instructions begin to execute, they have the required data, even if the loaded data changes.
- 128 integer registers; up to 96 rotating

  Note: 32 registers are fixed and 96 are "stacked". A procedure call can allocate up to 96 of the stacked registers and still has access to the 32 common registers. Each procedure has its own register frame, which is flexible in size. Since most procedure calls will allocate only a few new registers, many calls can be made before the physical limits of the register file are exceeded. A dedicated piece of hardware called the Register Stack Engine (RSE) will quickly and automatically spill older registers to free up space in the register stack for the new request. The RSE will also restore spilled registers as needed.
- 128 floating-point registers; up to 96 rotating
- 64 1-bit predicate registers; up to 48 rotating
- 8 branch registers
- 128 application registers (for example, loop or epilog counters for loop optimization)
- Performance Monitor Unit (PMU)
- Advanced Load Address Table (ALAT) ALAT keeps track of speculative, or advance loads. However, an excessive number of ALAT comparisons that result in a failed advance load will seriously degrade performance.
- 3 predicated instructions in a single 128-bit bundle
- 2 bundles (that is, 6 instructions) per clock cycle
- 6 integer units
- 2 loads and 2 stores per clock cycle
- 11 issue ports

Main Memory - Global Shared Memory

SGI Altix systems dramatically reduce the time and resources required to run applications by managing extremely large data sets in a single, system-wide, shared-memory space called global shared memory. Global shared memory means that a single memory address space is visible to all system resources, including microprocessors and I/O, across all nodes. Systems with global shared memory allow access to all data in the system's memory directly and efficiently, without having to move data through I/O or network bottlenecks. On the contrary, clusters with multiple nodes without global shared memory must pass copies of data, often in the form of messages, which can greatly complicate programming and slow down performance by increasing the time processors must wait for data.

If an Altix system is configured as a multi-partition cluster, global shared memory can be achieved by using a sophisticated system memory interconnect like SGI's NUMAlink and application libraries that enable shared-memory calls, such as MPT and XPMEM (a driver which allows shared memory across partitions) from SGI.

To configure an Altix system as a single system image machine, special versions of a scalable operation system from SGI is used and no XPMEM is needed. The current version of the OS used is "2.6.16.60-0.42.9.1-nasa64k #1 SMP".

The SGI Altix systems use the non-uniform memory access (NUMA) model. Memory subsystems from different nodes are connected through SHUB and NUMAlink interconnects.

Latency:

The local memory latency (within a node) is about 145 nanoseconds (ns). Latency from the other node of the same C-brick is 290 ns. Each additional router hop adds 45 - 50 ns (for NUMAlink 3 protocol). Each meter of NUMAlink cable adds 10 ns.

Maximum number of router hops:

- 16 CPUs - 3 hops
- 32 CPUs - 4 hops
- 64 CPUs - 5 hops
- 128 CPUs - 5 hops
- 256 CPUs - 7 hops

Bandwidth:

The Altix memory subsystem uses PC-style double data rate (DDR) SDRAM DIMMs. Each SHUB supports four DDR buses. Each DDR bus may contain up to four DIMMs. The four memory buses are independent and can operate simultaneously to provide up to 12.8 GB/sec of memory bandwidth. (Local memory bandwidth for DIMM type PC2700 is 10.2 GB/sec; and for type PC3200, it is 12.8 GB/sec.) While the local processor bus has a peak bandwidth (between L3 cache and memory) of 6.4 GB per second, the local memory subsystem has enough bandwidth to fully saturate the local processor demands while leaving available bandwidth to service remote processor and I/O memory requests.

Columbia Home Filesystems

DRAFT

This article is being reviewed for completeness and technical accuracy.

Columbia's home fileystem (/u/username) is NFS-mounted on the Columbia front-end (cfe2) and compute nodes (Columbia21-24).

Once a user is granted an account on Columbia, the home directory is set up automatically during his/her first login.

Quota and Policy

Disk space quota limits are enforced on the home filesystem. By default, the soft limit is 4GB and the hard limit is 5GB. There are no inode limits on the home filesystem.

To check your quota and usage on your home filesystem, do:

```
%quota -v
Disk quotas for user username (uid xxxx):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
  ch-rg1:/home6    4888  4000000 5000000            294       0       0
```

The quota policy for NAS states that if you exceed the soft quota, an email will be sent to inform you of your current usage and how much of your grace period remains. It is expected that a user will occasionally exceed their soft limit as needed; however after 14 days, users who are still over their soft limit will have their batch queue access to Pleiades disabled. If you believe that you have a long-term need for higher quota limits, you should send an email justification to support@nas.nasa.gov. This will be reviewed by the HECC Deputy Project Manager, Bill Thigpen, for approval.

The quota policy for NAS can be found here.

Backup Policy

Files on the home filesystem are backed up daily.

Columbia CXFS Filesystems

Columbia CXFS filesystems (/nobackup[1-2][a-i]) are shared and accessible from cfe2 and Columbia21-24. This allows user jobs to be load-balanced across Columbia's systems without forcing users to move their data to a particular Columbia system.

Users will have a nobackup directory on one of these shared file systems. To find out where your nobackup directory is, log in to the front-end node and type the following shell command:

```
cfe2% ls -d /nobackup[1-2][a-i]/$USER
/nobackup1f/username/
```

In this example, the user is assigned to /nobackup1f.

Default Quota and Policy on /nobackup

Disk space and inodes quotas are enforced on the CXFS /nobackup[1-2][a-i] filesystems. The default soft and hard limits for inodes are 25,000 and 50,000, respectively. Those for disk space are 200GB and 400GB, respectively. To check your disk space and inodes usage and quotas on your CXFS filesystem, do the following:

```
cfe2% quota -v
Disk quotas for user username (uid xxxx):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
/dev/cxvm/nobackup1f
                1673856  210000000 420000000              10973   25000   50000
```

The NAS quota policy states that if you exceed the soft quota, an email will be sent to inform you of your current usage and how much of your grace period remains. It is expected that users will occasionally exceed their soft limit, as needed; however after 14 days, users who are still over their soft limit will have their batch queue access to Columbia disabled.

If you anticipate having a long-term need for higher quota limits, please send a justification via email to support@nas.nasa.gov. This will be reviewed by the HECC Deputy Project Manager for approval.

For more information, see also, Quota Policy on Disk Space and Files.

Important: Backup Policy

As the names suggest, these filesystems are not backed up, so any files that are removed *cannot* be restored. Essential data should be stored on Lou1-3 or onto other more permanent storage.

Accessing CXFS from Lou

The Columbia CXFS filesystems are also mounted on Lou1-3. This allows you to copy files between the CXFS filesystems and your Lou home filesystem, using the *cp* or *cxfscp* commands on Lou.

Columbia Front-End Usage Guidelines

DRAFT

This article is being reviewed for completeness and technical accuracy.

The front-end system, cfe2, provide an environment that allows users to get quick turnaround while performing the following: file editing; file management; short debugging and testing sessions; and batch job submission to the compute systems.

Running long and/or large (in terms of memory and/or number of processors) debugging or production jobs interactively or in the background of cfe2 is considered to be inconsiderate behavior to the rest of the user community. If you need help submitting such jobs to the batch systems, please contact a NAS scientific consultant at (650) 604-4444 or 1-800-331-USER or send e-mail to: support@nas.nasa.gov

Jobs that cause significant impact on the system load of the Columbia front-end machine (cfe2) are candidates for removal in order to bring the front-end systems back to a normal and smooth environment for all users. A *cron* job regularly monitors the system load and determines if job removal is necessary. The criteria for job removal are described below. Owners of any removed jobs will receive a notification e-mail.

1. To be eligible for removal, the number of processors a front-end interactive job uses can be one (1) or more. Exceptions to this are those programs, utilities, etc. common to users and/or NASA missions that are listed in an "exception file". Examples of these would be:

   bash cp csh emacs gzip rsync scp sftp sh ssh tar tcsh

   Users can submit program names to be added to this exception file by mailing requests to: support@nas.nasa.gov
2. For qualifying processes, the CPU time usage of each process in a job has, on the average, exceeded a threshold defined as:

   (20 min x 8 / number of processes for the job)

   That is, a baseline for removal is a job with 8 processors running for more than 20 minutes. The maximum amount of time allowed for each processor in a job is scaled using the formula:

   20 min x 8 cpu / number-of-processes

   Therefore, the following variations are possible:

   - 160 minutes = (20 * 8) / 1 cpu

- ♦ 80 minutes = (20 * 8) / 2 cpu
- ♦ 40 minutes = (20 * 8) / 4 cpu
- ♦ 20 minutes = (20 * 8) / 8 cpu
- ♦ 10 minutes = (20 * 8) / 16 cpu
- ♦ 5 minutes = (20 * 8) / 32 cpu
- ♦ 2.5 minutes = (20 * 8) / 64 cpu

The conditions of removal are subject to change, when necessary.

Porting & Developing Applications

Porting & Developing: Overview

DRAFT

This article is being reviewed for completeness and technical accuracy.

When you are in the process of developing a code or porting a code from another platform, it is important that the code runs correctly and/or reproduces the results from another platform.

These are some steps you can follow when developing or porting a code or when testing a new version of a compiler.

General guidelines:

- Start with small problem sizes and a few time steps/iterations so that you won't have to wait in the queue for a long time just to check whether the program is running correctly. Setting up your PBS script, data files, and getting the program to run correctly can often be done with 10 minute jobs.
- Use PBS' debug queue to get better turn-around time (q=debug).
- While porting, make the fewest changes possible in the code.
- Use the same data sets to compare results on both old and new platforms.
- Don't assume that an absence of error messages means the program is running correctly on either the old or the new platforms.
- Be attentive to porting user data files. Fortran FORM='unformatted' files cannot be assumed to be portable.
- Don't assume that the new platform is wrong and the old platform is right. Both might be wrong.

Other useful information that helps you to port or develop a code on NAS HECC systems can be found in subsequent articles.

Endian and Related Environment Variables or Compiler Options

DRAFT

This article is being reviewed for completeness and technical accuracy.

Intel Fortran expects numeric data, both integer and floating-point data, to be in native little endian order, in which the least-significant, right-most zero bit (bit 0) or byte has a lower address than the most-significant, left-most bit (or byte).

If your program needs to read or write unformatted data files that are not in little endian order, you can use one of the six methods (listed in the order of precedence) provided by Intel below.

1. Set an environment variable for a specific unit number before the file is opened. The environment variable is named **FORT_CONVERTn**, where n is the unit number. For example:

   ```
   setenv FORT_CONVERT28 BIG_ENDIAN
   ```

   No source code modification or recompilation is needed.

2. Set an environment variable for a specific file name extension before the file is opened. The environment variable is named **FORT_CONVERT.ext** or **FORT_CONVERT_ext**, where ext is the file name extension (suffix). The following example specifies that a file with an extension of .dat is in big endian format:

   ```
   setenv FORT_CONVERT.DAT BIG_ENDIAN
   ```

   Some Linux command shells may not accept a dot (.) for environment variable names. In that case, use FORT_CONVERT_ext instead.

   No source code modification or recompilation is needed.

3. Set an environment variable for a set of units before any files are opened. The environment variable is named **F_UFMTENDIAN**.

   Syntax:

   Csh: setenv F_UFMTENDIAN MODE;EXCEPTION

   Sh : export F_UFMTENDIAN=MODE;EXCEPTION

MODE = big | little

EXCEPTION = big:ULIST | little:ULIST | ULIST

ULIST = U | ULIST,U

U = decimal | decimal-decimal

MODE defines the current format of the data, represented in the files; it can be omitted. The keyword "little" means that the data have little- endian format and will not be converted. For IA-32 systems, this keyword is a default. The keyword "big" means that the data have big endian format and will be converted. This keyword may be omitted together with the colon.

EXCEPTION is intended to define the list of exclusions for MODE; it can be omitted. EXCEPTION keyword (little or big) defines data format in the files that are connected to the units from the EXCEPTION list. This value overrides MODE value for the units listed.

Each list member U is a simple unit number or a number of units. The number of list members is limited to 64. decimal is a non-negative decimal number less than 2**32.

The environment variable value should be enclosed in quotes if the semicolon is present.

Converted data should have basic data types, or arrays of basic data types. Derived data types are disabled.

Example:

- setenv F_UFMTENDIAN big

  All input/output operations perform conversion from big-endian to little-endian on READ and from little-endian to big-endian on WRITE.

- setenv F_UFMTENDIAN "little;big:10,20"

  or setenv F_UFMTENDIAN big:10,20

  or setenv F_UFMTENDIAN 10,20

  In this case, only on unit numbers 10 and 20 the input/output operations perform big-little endian conversion.

- setenv F_UFMTENDIAN "big;little:8"

In this case, on unit number 8 no conversion operation occurs. On all other units, the input/output operations perform big-little endian conversion.

- ♦ setenv F_UFMTENDIAN 10-20

  Define 10, 11, 12, ...19, 20 units for conversion purposes; on these units, the input/output operations perform big-little endian conversion.

4. Specify the **CONVERT** keyword in the OPEN statement for a specific unit number. Note that a hard-coded OPEN statement CONVERT keyword value cannot be changed after compile time. The following OPEN statement specifies that the file graph3.dat is in VAXD unformatted format:

   ```
   OPEN (CONVERT='VAXD', FILE='graph3.dat', FORM='UNFORMATTED',
   UNIT=15)
   ```

5. Compile the program with an **OPTIONS** statement that specifies the CONVERT=keyword qualifier. This method affects all unit numbers using unformatted data specified by the program. For example, to use VAX F_floating and G_floating as the unformatted file format, specify the following OPTIONS statement:

   ```
   OPTIONS /CONVERT=VAXG
   ```

6. Compile the program with the command-line **-convert keyword** option, which affects all unit numbers that use unformatted data specified by the program. For example, the following command compiles program file.for to use VAXD floating-point data for all unit numbers:

   ```
   ifort file.for –o vconvert.exe –convert vaxd
   ```

In addition, if the record length of your unformatted data is in byte units (Intel Fortran default is in word units), use the **-assume byterecl** compiler option when compiling your source code.

OpenMP

DRAFT

This article is being reviewed for completeness and technical accuracy.

OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for various platforms.

Intel version 11.x compilers support OpenMP spec-3.0 while 10.x compilers support spec-2.5.

Building OpenMP Applications

The following Intel compiler options can be used for building or analyzing OpenMP applications:

- *-openmp*

  Enables the parallelizer to generate multithreaded code based on OpenMP directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems. The *-openmp* option works with both *-O0* (no optimization) and any optimization level of *-O*. Specifying *-O0* with *-openmp* helps to debug OpenMP applications.

  Note that setting *-openmp* also sets *-automatic*, which causes all local, non-SAVEd variables to be allocated to the run-time stack, which may provide a performance gain for your applications. However, if your program depends on variables having the same value as the last time the routine was invoked, your program may not function properly. If you want to cause variables to be placed in static memory, specify option *-save*. If you want only scalar variables of certain intrinsic types (integer, real, complex, logical) to be placed on the run-time stack, specify option *-auto-scalar*.
- *-assume cc_omp* or *-assume nocc_omp*

  -assume cc_omp enables conditional compilation as defined by the OpenMP Fortran API. That is, when "!$space" appears in free-form source or "c$spaces" appears in column 1 of fixed-form source, the rest of the line is accepted as a Fortran line.

  -assume nocc_omp tells the compiler that conditional compilation as defined by the OpenMP Fortran API is disabled unless option -openmp (Linux) or /Qopenmp (Windows) is specified.

- *-openmp-lib legacy* or *-openmp-lib compat*

  Choosing *-openmp-lib legacy* tells the compiler to use the legacy OpenMP run-time library (*libguide*). This setting does not provide compatibility with object files created using other compilers. This is the default for Intel version 10.x compilers.

Choosing *-openmp-lib compat* tells the compiler to use the compatibility OpenMP run-time library (*libiomp*). This is the default for Intel version 11.x compilers.

On Linux systems, the compatibility Intel OpenMP run-time library lets you combine OpenMP object files compiled with the GNUgcc or gfortran compilers with similar OpenMP object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

You cannot link object files generated by the Intel® Fortran compiler to object files compiled by the GNU Fortran compiler, regardless of the presence or absence of the *-openmp* (Linux) or /Qopenmp (Windows) compiler option. This is because the Fortran run-time libraries are incompatible.

NOTE: The compatibility OpenMP run-time library is not compatible with object files created using versions of the Intel compiler earlier than 10.0.

- *-openmp-link dynamic* or *-openmp-link static*

  Choosing *-openmp-link dynamic* tells the compiler to link to dynamic OpenMP run-time libraries. This is the default for Intel version 11.x compilers.

  Choosing *-openmp-link static* tells the compiler to link to static OpenMP run-time libraries.

  Note that the compiler options *-static-intel* and *-shared-intel* have no effect on which OpenMP run-time library is linked.

  Note that this option is only available for newer Intel compilers (version 11.x).
- *-openmp-profile*

  Enables analysis of OpenMP applications. To use this option, you must have Intel(R) Thread Profiler installed, which is one of the Intel(R) Threading Tools. If this threading tool is not installed, this option has no effect.

  Note that Intel Thread Profiler is not installed on Pleiades.
- *-openmp-report[n]*

  Controls the level of diagnostic messages of the OpenMP parallelizer. n=0,1,or 2.
- *-openmp-stub*

  Enables compilation of OpenMP programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.

OpenMP Environment Variables

There are a few OpenMP environment variables one can set. The most commonly used are:

- *OMP_NUM_THREADS num*

  Sets number of threads for parallel regions. Default is 1 on Pleiades. Note that you can use *ompthreads* in the PBS resource request to set values for OMP_NUM_THREADS. For example:

  ```
  %qsub -I -lselect=1:ncpus=4:ompthreads=4
  Job 991014.pbspl1.nas.nasa.gov started on Sun Sep 12 11:33:06 PDT 2010
  ...
  PBS r3i2n9> echo $OMP_NUM_THREADS
  4
  PBS r3i2n9>
  ```

- *OMP_SCHEDULE type[,chunk]*

  Sets the run-time schedule type and chunk size. Valid OpenMP schedule types are *static, dynamic, guided,* or *auto*. Chunk is a positive integer.

- *OMP_DYNAMIC true* or *OMP_DYNAMIC false*

  Enables or disables dynamic adjustment of threads to use for parallel regions.

- *OMP_STACKSIZE size*

  Specifies size of stack for threads created by the OpenMP implementation. Valid values for size (a positive integer) are *size, size*B*, size*K*, size*M*, size*G. If units B, K, M or G are not specified, size is measured in kilobytes (K).

  Note that this feature is included in OpenMP spec-3.0, but not in spec-2.5.

Note that Intel also provides a few additional environment variables. The most commonly used are:

- *KMP_AFFINITY type*

  Binds OpenMP threads to physical processors. Avaiable *type*: *compact, disabled, explicit, none, scatter*. For more information on the various types, see this Intel web page.

  `There is a conflict between` *`KMP_AFFINITY`* `in Intel 11.x runtime`

```
library and dplace, causing all threads to be placed on a
single CPU when both are used. It is recommended that
KMP_AFFINITY be set to disabled when using dplace.
```

- *KMP_MONITOR_STACKSIZE*

  Sets stacksize in bytes for monitor thread.

- *KMP_STACKSIZE*

  Sets stacksize in bytes for each thread.

For more information, please see the official OpenMP web site.

Compilers

Intel Compiler

DRAFT

This article is being reviewed for completeness and technical accuracy.

Intel compilers are recommended for building your applications on either Pleiades or Columbia.

On Columbia, a system default version has been loaded automatically. On Pleiades, there is no system default--you must load a specific module. Use the "module avail" command on Pleiades to see what versions are available and load an Intel compiler module before compiling. For example:

```
% module load comp-intel/11.1.072
```

Notice that when a compiler module is loaded, some environment variables, such as FPATH, INCLUDE, LD_LIBRARY_PATH, etc., are set or modified to add the paths to certain commands, include files, or libraries, to your environment. This helps to simplify the way you do your work.

To check what environmant variables will be modified for a module, do, for example:

```
% module show comp-intel/11.1.072
```

On Columbia and Pleiades, there are Intel compilers for both Fortran and C/C++:

- **Intel Fortran Compiler: ifort (version 8 and above)**

  The ifort command invokes the Intel(R) Fortran Compiler to preprocess, compile, assemble, and link Fortran programs.

  ```
  % ifort [options] file1 [file2 ...]
  ```

  Read **man ifort** for all available compiler options.

  To see the compiler options by categories, do:

  ```
  % ifort -help
  ```

  file*N* is a Fortran source (.f .for .ftn .f90 .fpp .F .FOR .F90 .i .i90), assembly (.s .S), object (.o), static library (.a), or other linkable file.

Source Files Suffix Interpretation:

- ♦ .f, .for, or .ftn : fixed-form source files
- ♦ .f90 : free-form F95/F90 source files
- ♦ .fpp, .F, .FOR, .FTN, or .FPP: fixed-form source files which must be preprocessed by the fpp preprocessor before being compiled
- ♦ .F90 : free-form Fortran source files which must be pre-pro- cessed by the fpp preprocessor before being compiled

- **Intel C/C++ compiler: icc and icpc (version 8 and above)**

The Intel(R) C++ Compiler is designed to process C and C++ programs on Intel-architecture-based systems. You can preprocess, compile, assemble, and link these programs.

```
% icc [options] file1 [file2 ...]
% icpc [options] file1 [file2 ...]
```

Read **man icc** for all available compiler options.

To see the compiler options by categories, do:

```
% icc -help
```

```
The icpc command uses the same compiler options as the icc
command. Invoking the compiler using icpc compiles .c, and .i
files as C++. Invoking the compiler using icc compiles .c and
.i files as C. Using icpc always links in C++ libraries. Using
icc only links in C++ libraries if C++ source is provided on
the command line.
```

file*N* represents a C/C++ source (.C .c .cc .cp .cpp .cxx .c++ .i), assembly (.s), object (.o), static library (.a), or other linkable file.

GNU Compiler Collection

DRAFT

This article is being reviewed for completeness and technical accuracy.

GCC stands for "GNU Compiler Collection". GCC is an integrated distribution of compilers for several major programming languages. These languages currently include C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada.

The GNU C and C++ compiler (gcc and g++) and Fortran compiler (gfortran) through the Linux OS distribution are available on Pleiades and Columbia. The current version installed (under /usr/bin) can be found with the following command:

```
% gcc -v
 ... gcc version 4.1.2 20070115 (SUSE Linux)
```

Newer versions of GNU compilers can be requested and installed as modules. Currently, there is a gcc/4.4.4 module, which includes gcc, g++, and gfortran, available on Pleiades.

Read **man gcc** and **man gfortran** for more information.

MPI Libraries

SGI MPT

DRAFT

This article is being reviewed for completeness and technical accuracy.

SGI's Message Passing Interface (MPI) is a component of the Message Passing Toolkit (MPT), which is a software package that supports parallel programming across a network of computer systems through a technique known as message passing. It requires the presence of an Array Services daemon (arrayd) on each host to run MPI processes.

SGI's MPT 1.x versions support the MPI 1.2 standard and certain features of MPI-2. The 2.x versions will be fully MPI-2 compliant.

On Columbia, the current system default version is mpt.1.16. A 2.x version will be available when the operating system is upgraded to SGI ProPack 7SP1.

On Pleiades, there is no default version. You can enable the recommended version, mpt.2.04.10789, by:

```
%module load mpi-sgi/mpt.2.04.10789
```

Note that certain environment variables are set or modified when an MPT module is loaded. To see what variables are set when a module is loaded (for example, mpi-sgi/mpt.2.04.10789), do:

```
%module show mpi-sgi/mpt.2.04.10789
```

To build an MPI application using SGI's MPT, use a command such as one of the following:

```
%ifort -o executable_name prog.f -lmpi
%icc -o executable_name prog.c -lmpi
%icpc -o executable_name prog.cxx -lmpi++ -lmpi
%gfortran -I/nasa/sgi/mpt/1.26/include -o executable_name prog.f -lmpi
%gcc  -o executable_name prog.c -lmpi
%g++ -o executable_name prog.cxx -lmpi++ -lmpi
```

MVAPICH

DRAFT

This article is being reviewed for completeness and technical accuracy.

MVAPICH is open source software developed largely by the Network-Based Computing Laboratory (NBCL) at Ohio State University. MVAPICH develops the Message Passing Interface (MPI) style of process-to-process communications for computing systems employing InfiniBand and other Remote Direct Memory Access (RDMA) interconnects.

MVAPICH software is typically used across the network of a cluster computer system for improved performance and scalability of applications.

MVAPICH is an MPI-1 implementation while MVAPICH2 is an MPI-2 implementation (conforming to MPI 2.2 standard) which includes all MPI-1 features.

MVAPICH1/MVAPICH2 are installed on Pleiades, but not Columbia. You must load in an MVAPICH1 or MVAPICH2 module before using it. For example:

```
%module load mpi-mvapich2/1.4.1/intel
```

A variety of MPI compilers, such as mpicc, mpicxx, mpiCC, mpif77, or mpif90, are provided in each MVAPICH/MVAPICH2 distribution. The correct compiler should be selected depending on the programming language of your MPI application.

To build an MPI application using MVAPICH1/MVAPICH2:

```
%mpif90 -o executable_name prog.f
%mpicc -o executable_name prog.c
```

Math & Scientific Libraries

MKL

DRAFT

This article is being reviewed for completeness and technical accuracy.

The Intel Math Kernel Library (MKL) is composed of highly optimized mathematical functions for engineering and scientific applications requiring high performance on Intel platforms. The functional areas of the library include linear algebra consisting of LAPACK and BLAS, fast Fourier transform (FFT), and vector transcendental functions.

MKL release 10.x is part of the Intel compiler 11.0 and 11.1 releases. Once you load in a 11.x compiler module, the path to the MKL library is automatically included in your default path. If you choose to use Intel compiler 10.x or earlier versions, you have to load an MKL module separately.

A Layered Model for MKL

Starting with MKL release 10.0, Intel employs a layered model for the MKL library. The layers are:

- Interface layer
    - LP64 interface (uses 32-bit integer type) or ILP64 interface (uses 64-bit integer type)
    - SP2DP interface

      which supports Cray-style naming in applications targeted for the Intel 64 or IA-64 architecture and using the ILP64 interface. SP2DP interface provides a mapping between single-precision names (for both real and complex types) in the application and double-precision names in Intel MKL BLAS and LAPACK.
- Threading layer
    - sequential

      The sequential (non-threaded) mode requires no Compatibility OpenMP* or Legacy OpenMP* run-time library, and does not respond to the environment variable OMP_NUM_THREADS or its Intel MKL equivalents. In this mode, Intel MKL runs unthreaded code. However, it is thread-safe, which means that you can use it in a parallel region from your own OpenMP code. You should use the library in the sequential mode only if you have a particular reason not to use Intel MKL threading. The sequential mode may be helpful when using Intel MKL with programs threaded with some non-Intel compilers or in other situations where you may, for various reasons, need a non-threaded version of the library (for instance, in some MPI cases).

  Note that the *sequential.* library depends on the POSIX threads library (pthread), which is used to make the Intel MKL software thread-safe and should be listed on the link line.

  - threaded

    The *threaded* library in MKL version 10.x supports the implementation of OpenMP that many compilers (Intel, PGI, GNU) provide.

- Computational layer

  For any given processor architecture (IA-32, IA-64, or Intel(R) 64) and OS, this layer has only one computational library to link with, regardless of the Interface and Threading layer.

- Compiler Support Run-time libraries

  - libiomp

    Intel(R) Compatibility OpenMP run-time library

  - libguide

    Intel(R) Legacy OpenMP run-time library

For example, to do a dynamic linking of myprog.f and parallel Intel MKL supporting LP64 interface, use:

```
ifort myprog.f -Wl,--start-group -lmkl_intel_lp64 \
-lmkl_intel_thread -lmkl_core -Wl,--end-group -openmp
```

If you are unsure of what MKL libraries to link with, use the suggestion provided in this Intel web site by providing the proper OS (e.g. Linux), processor architecture (e.g. Intel(R) 64), compiler (e.g. Intel or Intel Compatible), dynamic or static linking, integer length, sequential or multi-threaded, OpenMP library, cluster library (e.g. BLACS, ScaLAPACK), MPI library (Intel MPI, MPICH2, SGIMPT, etc.).

The -mkl Switch of Intel Compiler Version 11.1

Starting from Intel compiler version 11.1, a -mkl switch is provided to link to certain parts of the MKL library.

```
-mkl[=]
```

```
        link to the Intel(R) Math Kernel Library (Intel(R) MKL) and
        bring in the associated headers
          parallel   - link using the threaded Intel(R) MKL libraries.
                       This is the default when -mkl is specified
          sequential - link using the non-threaded Intel(R) MKL libraries
          cluster    - link using the Intel(R) MKL Cluster libraries plus
                       the sequential Intel(R) MKL libraries
```

The libraries that are linked in for

```
  * -mkl=parallel

        --start-group \
        -lmkl_solver_lp64 \
        -lmkl_intel_lp64 \
        -lmkl_intel_thread \
        -lmkl_core \
        -liomp5 \
        --end-group \

  * -mkl=sequential

        --start-group \
        -lmkl_solver_lp64_sequential \
        -lmkl_intel_lp64 \
        -lmkl_sequential \
        -lmkl_core \
        --end-group \

  * -mkl=cluster

        --start-group \
        -lmkl_solver_lp64 \
        -lmkl_intel_lp64 \
        -lmkl_cdft_core \
        -lmkl_scalapack_lp64 \
        -lmkl_blacs_lp64 \
        -lmkl_sequential \
        -lmkl_core \
        -liomp5 \
        --end-group \
```

Where to find more information about MKL

Man pages and two PDF files from Intel are available for each version of MKL.

- Man pages of Intel MKL

  A collection of man pages of Intel MKL functions are available under the man3 subdirectory (e.g., /nasa/intel/Compiler/11.1/072/man/en_US/man3) of the MKL installation. You will have to load an MKL module or an Intel compiler 11.x module before you can see the man pages. For example,

```
% module load comp-intel/11.1.072
% man gemm
```

provides information about [s,d,c,z,sc,dz]gemm routines.

Unfortunately, there does not appear to be a 'man mkl' page.

- Intel MKL Reference Manual (mklman.pdf)

  Contains detailed descriptions of the functions and interfaces for all library domains:

  - ♦ BLAS
  - ♦ LAPACK
  - ♦ ScaLAPACK
  - ♦ Sparse Solver
  - ♦ Interval Linear Solvers
  - ♦ Vector Math Library (VML)
  - ♦ Vector Statistical Library (VSL)
  - ♦ Conventional DFTs and Cluster DFTs
  - ♦ Partial Differential Equations support
  - ♦ Optimization Solvers

- Intel MKL User's Guide (userguide.pdf)

  Provides Intel MKL usage information in greater detail:

  - ♦ getting started information
  - ♦ application compiling and linking depending on a particular platform and function domain
  - ♦ building custom DLLs
  - ♦ configuring the development environment
  - ♦ coding mixed-language calls
  - ♦ threading
  - ♦ memory management
  - ♦ ways to obtain best performance

The two pdf files can be found in the 'doc' or 'Documentation' directory of the MKL installation. For example, on Pleiades,

- ♦ MKL version 10.0.011

  /nasa/intel/mkl/10.0.011/doc

- ♦ The version included in the Intel compiler module 11.1.072

  /nasa/intel/Compiler/11.1/072/Documentation/en_US/mkl

SCSL

DRAFT

This article is being reviewed for completeness and technical accuracy.

SCSL is a comprehensive collection of scientific and mathematical functions that have been optimized for use on the Altix systems such as Columbia . The libraries include optimization of basic linear algebra subprograms (BLAS), a linear algebra package, signal processing functions such as fast Fourier transforms (FFTs), and liner filtering operations and other basic solver functions. More information can be found through 'man scsl'.

```
Starting with ProPack 5, SCSL is no longer supported by SGI.
Although SCSL is still available on Columbia (but not on Pleiades),
users are recommended to use Intel MKL instead.
```

SCSL version(s) available on Columbia systems:

- scsl.1.5.0.0 (does not work properly with intel-comp.9.1.039)
- scsl.1.5.1.0
- scsl.1.5.1.1 (contains Scalapack in libsdsm.so)
- scsl.1.6.1.0

To use SCSL, link one of the following libraries:

```
-lscs
-lscs_mp      (for multi-threaded programs)
-lscs_i8
-lscs_i8_mp
```

MKL FFTW Interface

DRAFT

This article is being reviewed for completeness and technical accuracy.

Some users have installed the FFTW library in their own directory (for example, /u/user/bin/fftw) and would link to the FFTW library as follows:

```
ifort -O3
  -I/u/user/bin/fftw/include   \
  -o fftw_xmpl.exe fftw_xmpl.f \
  -L/u/user/bin/fftw/lib -lfftw3
```

An MKL FFTW interface has been created for Intel compiler version 11.0.083 and later versions. Users no longer have to keep their own copy of FFTW. Follow these steps to use the MKL FFTW interface:

- Load a compiler module 11.0.083 or a later version such as comp-intel/11.1.072

  ```
  module load comp-intel/11.1.072
  ```

- Compile and link

  ```
  ifort -O3                                            \
    -I/nasa/intel/Compiler/11.1/072/mkl/include/fftw    \
    -o fftw_xmpl.exe fftw_xmpl.f                        \
    -lfftw3xf_intel -lmkl_intel_lp64 -lmkl_intel_thread \
    -lmkl_core -lguide
  ```

Program Development Tools

Recommended Intel Compiler Debugging Options

DRAFT

This article is being reviewed for completeness and technical accuracy.

- Commonly used options for debugging:

  -O0
  : Disables optimizations. Default is -O2

  -g
  : Produces symbolic debug information in object file (implies -O0 when another optimization option is not explicitly set)

  -traceback
  : Tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run-time.

    ```
    Specifying -traceback will increase the size of the
    executable program, but has no impact on run-time
    execution speeds.
    ```

  -check all
  : Checks for all run-time failures. **Fortran only.**

  -check bounds
  : Alternate syntax: -CB. Generates code to perform run-time checks on array subscript and character substring expressions. **Fortran only.**

    ```
    Once the program is debugged, omit this option to reduce
    executable program size and slightly improve run-time
    performance.
    ```

  -check uninit
  : Checks for uninitialized **scalar** varaibles without the SAVE attribute. **Fortran only.**

  -check-uninit
  : Enables run-time checking for uninitialized variables. If a variable is read before it is written, a run-time error routine will be called. Run-time checking of undefined variables is only implemented on local, scalar variables. It is not

implemented on dynamically allocated variables, extern variables or static variables. It is not implemented on structs, classes, unions or arrays. **C/C++ only**.

-ftrapuv

Traps uninitialized variables by setting any uninitialized local variables that are allocated on the stack to a value that is typically interpreted as a very large integer or an invalid address. References to these variables are then likely to cause run-time errors that can help you detect coding errors. This option sets -g.

-debug all

Enables debug information and control output of enhanced debug information. To use this option, you must also specify the -g option.

-gen-interfaces -warn interfaces

Tells the compiler to generate an interface block for each routine in a source file; the interface block is then checked with -warn interfaces

- Options for handling floating-point exceptions:

-fpe{0|1|3}

Allows some control over floating-point exception (divide by zero, overflow, invalid operation, underflow, denormalized number, positive infinity, negative infinity or a NaN) handling for the **main program** at run-time. **Fortran only.**

· -fpe0: underflow gives 0.0; abort on other IEEE exceptions
· -fpe3: produce NaN, signed infinities, and denormal results

Default is -fpe3 with which all floating-point exceptions are disabled and floating-point underflow is gradual, unless you explicitly specify a compiler option that enables flush-to-zero. Use of -fpe3 on IA-64 systems such as Columbia will slow run-time performance.

-fpe-all={0|1|3}

Allows some control over floating-point exception handling for **each routine** in a program at run-time. Also sets -assume ieee_fpe_flags. Default is -fpe-all=3. **Fortran only.**

-assume ieee_fpe_flags

Tells the compiler to save floating-point exception and status flags on routine entry and restore them on routine exit. This option can slow runtime performance. **Fortran only.**

-ftz

Flushes denormal results to zero when the application is in the gradual underflow mode. This option has effect only when compiling the **main program**. It may improve performance if the denormal values are not critical to your application's behavior. For IA-64 systems (such as Columbia), -O3 sets -ftz. For Intel 64 systems (such as Pleiades), every optimization option O level, except -O0, sets -ftz.

- Options for handling floating-point precision:

  -mp
  : Enables improved floating-point consistency during calculations. This option limits floating-point optimizations and maintains declared precision. -mp1 restricts floating-point precision to be closer to declared precision. It has some impact on speed, but less than the impact of -mp.

  -fp-model precise
  : Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations. These semantics ensure the accuracy of floating-point computations, but they may slow performance.

  -fp-model strict
  : Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. This is the strictest floating-point model.

  -fp-speculation=off
  : Disables speculation of floating-point operations. Default is -fp-speculation=fast

  -pc{64|80}
  : For Intel EM64 only. Some floating-point algorithms are sensitive to the accuracy of the significand, or fractional part of the floating-point value. For example, iterative operations like division and finding the square root can run faster if you lower the precision with the -pc option. -pc64 sets internal FPU precision to 53-bit significand. -pc80 is the default and it sets internal FPU precision to 64-bit significand.

Totalview

DRAFT

This article is being reviewed for completeness and technical accuracy.

TotalView is a GUI-based debugging tool that gives you control over processes and thread execution and visibility into program state and variables for C, C++ and Fortran applications. It also provides memory debugging to detect errors such as memory leaks, deadlocks and race conditions, etc.

Totalview allows you to debug serial, OpenMP, or MPI codes.

Totalview is available on both Pleiades and Columbia. See Totalview Debugging on Pleiades for some basic instructions on how to start using Totalview on Pleiades.

See Totalview Debugging on Columbia for some basic instructions on how to start using Totalview on Columbia.

Totalview Debugging on Pleiades

DRAFT

This article is being reviewed for completeness and technical accuracy.

TotalView is an advanced debugger for complex and parallel codes. Its versions have been installed as modules. To find out what versions of totalview are available, use the 'module avail' command.

There are additional steps needed in order to start the TotalView GUI. You'll need to rely on the ForwardX11 feature of your ssh. First, you'll have to make sure that your sysadmin had turned on ForwardX11 when SSH was installed on your system or use the -X or -Y (if available) options of ssh to enable X11 forwarding for your SSH session.

For debugging on a back-end node, do:

- Compile your code with -g
- Start a PBS session. For example:

  ```
  % qsub -I -V -lselect=2:ncpus=8,walltime=1:00:00
  ```
- Test the X11 forwarding with xlock

  ```
  % xclock
  ```
- Load the totalview module

  ```
  % module load apps/etnus/totalview.8.6.2-1
  ```
- Set the environment variable TOTALVIEW

  ```
  % setenv TOTALVIEW `which totalview` (for csh users)
  ```
  or
  ```
  % export TOTALVIEW=`which totalview` (for bash users)
  ```
- Start TotalView debugging
  - ♦ For serial applications:
    - ◊ Simply start totalview with your application as an argument

      ```
      % totalview ./a.out
      ```

If your application requires arguments:

```
% totalview ./a.out -a app_arg_1 app_arg_2
```

- For MPI applications:

1. Make sure you load the appropriate modules, including the compiler, and mpi module. For example:

   For applications built with SGI's MPT, make sure that you have loaded the latest MPT module:

   ```
   % module load comp-intel/11.1.072
   % module load mpi-sgi/mpt.1.26
   ```

   For applications built with MVAPICH:

   ```
   % module load comp-intel/11.1.072
   % module load mpi-mvapich2/1.4.1/intel
   ```
2. Launch totalview by typing "totalview" all by itself. Once the totalview windows pop up, you will see four tabs in the "New Program" window: Program, Arguments, Standard I/O and Parallel.

3. Fill in the executable name in the "Program" box or use the Browse button to find the executable

4. Give any arguments to your executable by clicking on the "Arguments" tab and filling in what you need. If you need to redirect input from a file, do so by clicking the "Standard I/O" tab and filling in what you need.

5. In the "Parallel" tab, select the parallel system option MVAPICH2 or mpt_1.26 depending on which version of MPI you have compiled with.

6. Enter in the number of processes in the 'tasks' box; leave the 'nodes' field 0. For example, if you run your application with 2 nodes x 4 MPI processes/node = 8 processes in total, fill in 8 in the 'tasks' box and 0 in the 'node' box.

7. Then press "Go" to start. Note that it may initially dump you into the mpiexec assembler source which is not your own code.

8. Respond to the popup dialog box which says "Process xxx is a parallel job. Do you want to stop the job now?" Choose "No" if you just want to run your application. Choose "Yes" if you want to set breakpoint in your source code or do other tasks before running.

More information about TotalView can be found at the Totalview online documentation website.

Totalview Debugging on Columbia

DRAFT

This article is being reviewed for completeness and technical accuracy.

TotalView is an advanced debugger for complex and parallel codes. It has been installed as modules. To find out what versions of totalview are available, use the command 'module avail totalview'.

You'll need to rely on the ForwardX11 feature of your ssh. First, you'll have to make sure that your sysadmin had turned on ForwardX11 when SSH was installed on your local system or use the -X or -Y (if available) options of ssh to enable X11 forwarding for your SSH session.

For debugging on the front-end cfe2, do:

- Login to the front-end cfe2
- Compile your code with -g
- Make sure that X11 forwarding works and test it with xclock

```
cfe2%echo $DISPLAY
cfe2:xx.0
cfe2%xclock
```

- Load the totalview module

```
cfe2% module load totalview.8.9.0-1
```

- Start totalview. For serial jobs:

  cfe2% totalview a.out

  For MPI jobs built with SGI's MPT library:

  cfe2% totalview mpirun.real -a -np xxx a.out

For debugging on a back-end node, do:

- Compile your code with -g
- Start a PBS session and pass in the environment variable DISPLAY. Assuming PBS assign your job to run on Columbia21

```
cfe2% qsub -I -v DISPLAY -lncpus=8,walltime=1:00:00
```

- Test the X11 forwarding with xlock

```
PBS(8cpus)columbia21% xclock
```

- Load the totalview module

```
PBS(8cpus)columbia21% module load totalview.8.9.0-1
```

- Start totalview. For serial jobs:

  PBS(8cpus)columbia21% totalview a.out

  For MPI jobs built with SGI's MPT library:

  PBS(8cpus)columbia21% totalview mpirun.real -a -np xxx a.out

More information about TotalView can be found at the Totalview online documentation website.

IDB

DRAFT

This article is being reviewed for completeness and technical accuracy.

The Intel Debugger is a symbolic source code debugger that debugs programs compiled by the Intel Fortran and C/C++ Compiler, and the GNU compilers (gcc, g++).

IDB is included in the Intel compiler distribution. For IA-64 systems such as Columbia, both the Intel 10.x and 11.x compiler distributions provide only an IDB command-line interface. To use IDB on Columbia, load an Intel 10.x or 11.x compiler module. For example:

```
%module load intel-comp.11.1.072
%idb
(idb)
```

For Intel 64 systems such as Pleiades, a command-line interface is provided in the 10.x distribution and is invoked with the command *idb* just like on Columbia. For the Intel 11.x compilers, both a graphical user interface (GUI), which requires a Java Runtime, and a command-line interface are provided. The command *idb* invokes the GUI interface by default. To use the command-line interface under 11.x compilers, use the command *idbc*. For example:

```
%module load comp-intel/11.1.072 jvm/jre1.6.0_20
%idb
.... This will bring up an IDB GUI ....

%module load comp-intel/11.1.072
%idbc
(idb)
```

Be sure to compile your code with the *-g* option for symbolic debugging.

Depending on the Intel compiler distributions, the Intel Debugger can operate in either the gdb mode, dbx mode or idb mode. The available commands under these modes are different.

For information on IDB in the 10.x distribution, read **man idb**.

For information on IDB in the 11.x distribution, read documentations under *pfe or cfe2:/nasa/intel/Compiler/11.1/072/Documentation/en_US/idb*

GDB

DRAFT

This article is being reviewed for completeness and technical accuracy.

The GNU Debugger, GDB, is available on both Pleiades and Columbia under /usr/bin. It can be used to debug programs written in C, C++, Fortran and Modula-a.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

Be sure to compile your code with *-g* for symbolic debugging.

GDB is typically used in the following ways:

- Start the debugger by itself
  ```
  %gdb
  (gdb)
  ```
- Start the debugger and specify the executable
  ```
  %gdb your_executable
  (gdb)
  ```
- Start the debugger, and specify the executable and core file
  ```
  %gdb your_executable core-file
  (gdb)
  ```
- Attach gdb to a running process
  ```
  %gdb your_executable pid
  (gdb)
  ```

At the prompt (gdb), type in commands such as *break* for setting a breakpoint, *run* for starting to run your executable, *step* for stepping into next line, etc. Read **man gdb** to learn more on using gdb.

Using pdsh_gdb for Debugging Pleiades PBS Jobs

DRAFT

This article is being reviewed for completeness and technical accuracy.

A script called *pdsh_gdb*, created by NAS staff Steve Heistand, is available on Pleiades under */u/scicon/tools/bin* for debugging PBS jobs **while the job is running**.

Launching this script from a Pleiades front-end node allows one to connect to each compute node of a PBS job and create a stack trace of each process. The aggregated stack trace from each process will be written to a user specified directory (by default, it is written to ~/tmp).

Here is an example of how to use this script:

```
pfe1% mkdir tmp
pfe1% /u/scicon/tools/bin/pdsh_gdb -j jobid -d tmp -s -u nas_username
```

More usage information can be found by launching pdsh_gdb without any option:

```
pfe1% /u/scicon/tools/bin/pdsh_gdb
```

Porting to Pleiades

Recommended compiler options

DRAFT

This article is being reviewed for completeness and technical accuracy.

Intel compiler versions 10.0, 10.1, 11.0, 11.1, and 12.0 are available on Pleiades as modules. Use the 'module avail' command to find available versions. Since NAS does not set a default version for users on Pleiades, be sure to use the 'module load ...' command to load the version you want to use.

In addition to the few flags mentioned in the article Recommended Intel Compiler Debugging Options, here are a few more to keep in mind:

Turn on optimization: *-O3*

If you do not specify an optimization level (*-On*, *n=0,1,2,3*), the default is *-O2*. If you want more aggressive optimizations, you can use *-O3*. Note that using *-O3* may not improve performance for some programs.

Generate optimized code for a processor type: *-xS, -xSSE4.1* or *-xSSE4.2*

Intel version 10.x, 11.x and 12.x compilers provide flags for generating optimized codes specialized for various instruction sets used in specific processors or microarchitectures.

Processor Type	Intel V10.x	Intel V11.x and above
Harpertown	-xS	-xSSE4.1
Nehalem-EP **Westmere-EP**	N/A	-xSSE4.2

Since the instruction set is upward compatible, an application which is compiled with *-xSSE4.1* can run on either Harpertown or Nehalem-EP or Westmere-EP processors. An application which is compiled with *-xSSE4.2* can run ONLY on Nehalem-EP and Westmere-EP processors.

If your goal is to get the best performance out of the Nehalem-EP/Westmere-EP processors, it is recommended that you do the following:

- Use either Intel 11.x or 12.x compilers as they are designed for Nehalem-EP/Westmere-EP micro-architecture optimizations.

- Use the Nehalem-EP/Westmere-EP processor specific optimization flag *-xSSE4.2*

  ```
  Warning: Running an executable built with the -xSSE4.2 flag on
  the Harpertown processors will result in the following error:
  ```

  Fatal Error: This program was not built to run on the processor in your system. The allowed processors are: Intel(R) processors with SSE4.2 and POPCNT instructions support.

If your goal is to have a portable executable that can run on either Harpertown or Nehalem-EP or Westmere-EP, you can choose one of the following approaches:

- use none of the above flags
- use *-xSSE4.1* (with version 11.x and 12.x compilers)
- use *-O3 -ipo -axSSE4.2,xSSE4.1*(with version 11.x and 12.x compilers).

  This allows a single executable that will run on any of the three Pleiades processor types with suitable optimization to be determined at run time.

Turn inlining on: *-ip* or *-ipo*

Use of *-ip* enables additional interprocedural optimizations for single file compilation. One of these optimizations enables the compiler to perform inline function expansion for calls to functions defined within the current source file.

Use of *-ipo* enables multifile interprocedural (IP) optimizations (between files). When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

Use a specific memory model: *-mcmodel=medium* and *-shared-intel*

Should you get a link time error relating to R_X86_64_PC32, add in the compiler option of *-mcmodel=medium* and the link option of *-shared-intel*. This happens if a common block is > 2gb in size.

Turn off all warning messages: *-w -vec-report0 -opt-report0*

Use of *-w* disables all warnings; *-vec-report0* disables printing of vectorizer diagnostic information; and *-opt-report0* disables printing of optimization reports.

Parallelize your code: *-openmp* or *-parallel*

-openmp handles OMP directives and *-parallel* looks for loops to parallelize.

For more compiler/linker options, read **man ifort, man icc**, or

```
%ifort -help
%icc -help
```

With SGI's MPT

DRAFT

This article is being reviewed for completeness and technical accuracy.

Among the many MPI libraries installed on Pleiades, it is recommended that you start with SGI's MPT library.

The available SGI MPT modules are:

```
mpi/mpt.1.25
mpi-sgi/mpt.1.26
mpi-sgi/mpt.2.04.10789
```

There is no default MPT version set, but you are recommended to start with the MPT 2.04.10789 version by loading the *mpi-sgi/mpt.2.04.10789* module. You should load the same module when you build your application on the front-end node and also inside your PBS script for running on the back-end nodes.

Note: Pleiades uses an InfiniBand (IB) network for interprocess RDMA (remote direct memory access) communications and there are two InfiniBand fabrics, designated as *ib0* and *ib1*. In order to maximize performance, SGI advises that the *ib0* fabric be used for all MPI traffic. The *ib1* fabric is reserved for storage related traffic. The default configuration for MPI is to use only the *ib0* fabric.

Environment Variables

When you load an MPT module, several paths (such as CPATH, C_INCLUDE_PATH, LD_LIBRARY_PATH, etc) and MPT or ARRAYD related variables are set or modified. For example, with the mpi-sgi/mpt.2.04.10789 module, the following MPT and ARRAYD related variables are reset to some non-default values:

```
setenv        MPI_BUFS_PER_HOST 256
setenv        MPI_IB_TIMEOUT 20
setenv        MPI_IB_RAILS 2
setenv        MPI_DSM_DISTRIBUTE 0 (for Harpertown processors)
setenv        MPI_DSM_DISTRIBUTE 1 (for Nehalem-EP and Westmere-EP processors)
setenv        ARRAYD_CONNECTTO 15
setenv        ARRAYD_TIMEOUT 180
```

The meanings of these variables and their default values are:

- MPI_BUFS_PER_HOST

  Determines the number of shared message buffers (16 KB each) that MPI is to allocate for each host (i.e., Pleiades node used in the run). These buffers are used to send and receive long inter-host messages.

Default: 96 pages (1 page = 16KB)

- MPI_IB_TIMEOUT

  When an IB card sends a packet it waits some amount of time for an ACK packet to be returned by the receiving IB card. If it does not receive one it sends the packet again. This variable controls that wait period. The time spent is equal to 4 * 2 ^ MPI_IB_TIMEOUT microseconds.

  Default: 18
- MPI_IB_RAILS

  If the MPI library uses the IB driver as the inter-host interconnect it will by default use a single IB fabric. If this is set to 2, the library will try to make use of multiple available separate IB fabrics (e.g. *ib0* and *ib1*) and split its traffic across them. If the fabrics do not have unique subnet IDs then the rail-config utility is required to have been run by the system administrator to enable the library to correctly use the separate fabrics.

  Default: 1

- MPI_DSM_DISTRIBUTE

  Activates NUMA job placement mode. This mode ensures that each MPI process gets a unique CPU and physical memory on the node with which that CPU is associated. This feature can also be overridden by using *dplace* or *omplace*. This feature is most useful if running on a dedicated system or running within a cpuset.

  Default: Enabled for MPT.1.26; Not Enabled for MPT.1.25

- ARRAYD_CONNECTTO

  Tuning this variable is useful when you want to run jobs through arrayd across a large cluster, and there is network congestion. Setting this variable to a higher value might slow down some array commands when a host is unavailable but it will help to prevent MPI start up problems due to connection time-out.

  Default: 5 seconds

- ARRAYD_TIMEOUT

  Tuning this variable is useful when you want to run jobs through arrayd across a large cluster, and there is network congestion. Setting this variable to a higher value might slow down some array commands when a host is unavailable but it will help to prevent MPI start up problems due to connection time-out.

Default: 45 seconds

For more MPT related variables, read **man mpi** after loading an MPT module. Some of them may be useful for some applications or for debugging purposes on Pleides. Here are a few of them for you to consider:

- MPI_BUFS_PER_PROC

  Determines the number of private message buffers (16 KB each) that MPI is to allocate for each process (i.e. MPI rank). These buffers are used to send long messages and intrahost messages.

  Default: 32 pages (1 page = 16KB)

- MPI_IB_FAILOVER

  When the MPI library uses IB and a connection error is detected, the library will handle the error and restart the connection a number of times equal to the value of this variable. Once there are no more failover attempts left and a connection error occurs, the application will be aborted.

  Default: 4

- MPI_COREDUMP

  Controls which ranks of an MPI job can dump core on receipt of a core-dumping signal. Valid values are *NONE, FIRST, ALL*, or *INHIBIT. NONE* means that no rank should dump core. *FIRST* means that the first rank on each host to receive a core-dumping signal should dump core. *ALL* means that all ranks should dump core if they receive a core-dumping signal. *INHIBIT* disables MPI signal-handler registration for core- dumping signals.

  Default: FIRST

- MPI_STATS (toggle)

  Enables printing of MPI internal statistics. Each MPI process prints statistics about the amount of data sent with MPI calls during the MPI_Finalize process.

  Default: Not enabled

- MPI_DISPLAY_SETTING

  If set, MPT will display the default and current settings of the environmental variables controlling it.

Default: Not enabled

- MPI_VERBOSE

  Setting this variable causes MPT to display information such as what interconnect devices are being used and environmental variables have been set by the user to non-default values. Setting this variable is equivalent to passing mpirun the -v option.

  Default: Not enabled

Building Applications

Building MPI applications with SGI's MPT library simply requires linking with -lmpi and/or -lmpi++. See the article SGI MPT for some examples.

Running Applications

```
MPI executables built with SGI's MPT are not allowed to run on the
Pleiades front-end nodes.
```

You can run your MPI job on the back-end nodes in an interactive PBS session or through a PBS batch job. After loading an MPT module, use **mpiexec**, not mpirun, to start your MPI processes. For example:

```
#PBS -lselect=2:ncpus=8:mpiprocs=4:model=har
....
module load mpi-sgi/mpt.2.04.10789
mpiexec -np N ./your_executable
```

The -np flag (with *N* MPI processes) can be omitted if the value of *N* is the same as the product of the value specified for *select* and that specified for *mpiprocs*.

Performance Issues

On Nehalem-EP and Westmere-EP nodes, if your MPI job uses all the processors in each node (i.e, 8 MPI processes/node for Nehalem-EP and 12 MPI processes/node for Westmere-EP), pinning MPI processes greatly helps the performance of the code. SGI's mpi-sgi/mpt.2.04.10789 will pin processes by default by setting the environment variable MPI_DSM_DISTRIBUTE to 1 (or true) when jobs are run on the Nehalem or Westmere nodes. On Harpertown nodes, setting MPI_DSM_DISTRIBUTE to 1 is not recommended due to a processor labeling issue.

If your MPI job do not use all the processors in each node, it is recommended that you disable MPI_DSM_DISTRIBUTE by

```
setenv MPI_DSM_DISTRIBUTE 0
```

and let the Linux kernel decide where to place your MPI processes. If you want to pin processes explicitly, you can use *dplace*. Beware that with SGI's MPT, only 1 shepherd process is created for the entire pool of MPI processes and the proper way of pinning using *dplace* is to skip the shepherd process. In addition, knowledge of the processor labeling in each processor type is essential when you use *dplace*. Below are the recommended ways of pinning an 8 MPI process job with every 4 processes on 4 processor cores of a node:

- Harpertown

```
mpiexec -np 8 dplace -s1 -c2,3,6,7 ./your_executable
```

- Nehalem-EP

```
mpiexec -np 8 dplace -s1 -c2,3,4,5 ./your_executable
```

- Westmere-EP

```
mpiexec -np 8 dplace -s1 -c4,5,6,7 ./your_executable
```

Further information about pinning can be found here.

With MVAPICH

DRAFT

This article is being reviewed for completeness and technical accuracy.

On Pleiades, there are multiple modules of MVAPICH2 built with either gcc or Intel compilers.

```
mpi-mvapich2/1.2p1/gcc
mpi-mvapich2/1.2p1/intel
mpi-mvapich2/1.2p1/intel-PIC
mpi-mvapich2/1.4.1/gcc
mpi-mvapich2/1.4.1/intel
```

The module *mpi-mvapich2/1.2p1/intel-PIC* was built with the -fpic compiler flag.

Building Applications

Here is an example of how to build an MPI application with MVAPICH2:

```
%module load mpi-mvapich2/1.4.1/intel
%module load comp-intel/11.1.072
%mpif90 program.f90
```

Running Applications

To run your job, submit your job through PBS. Within the PBS script, there are two ways to run MPI applications built with MVAPICH2.

1.
   ```
   #PBS ..
   ...
   module load mpi-mvapich2/1.4.1/intel
   module load comp-intel/11.1.072

   mpiexec -np TOTAL_CPUS your_executable
   ```

2.
   ```
   #PBS ..
   ...
   module load mpi-mvapich2/1.4.1/intel
   module load comp-intel/11.1.072

   mpirun_rsh -np TOTAL_CPUS -hostfile $PBS_NODEFILE your_executable
   ```

Performance Issues

To pin processes, the MVAPICH library uses the environment variable VIADEV_USE_AFFINITY, which does something similar to SGI's MPI_DSM_DISTRIBUTE.

By default, VIADEV_USE_AFFINITY is set to 1.

If you wish to pin processes explicitly, beware that with MVAPICH, 1 shepherd process is created for each MPI process. You can use the command

```
/u/scicon/tools/bin/qsh.pl jobid \
  'ps -C executable -L -opsr,pid,ppid,lwp,time,comm'
```

to see these processes of your running job. To properly pin MPI processes using *dplace*, one cannot skip the shepherd processes. In addition, knowledge of the processor labeling in each processor type is essential when you use *dplace*. Below are the recommended ways of pinning an 8 MPI process job with every 4 processes on 4 processors of a node:

- Harpertown

  ```
  mpiexec -np 8 dplace -c2,3,6,7 ./your_executable
  ```

- Nehalem-EP

  ```
  mpiexec -np 8 dplace -c2,3,4,5 ./your_executable
  ```

- Westmere-EP

  ```
  mpiexec -np 8 dplace -c4,5,6,7 ./your_executable
  ```

Further information about pinning can be found here.

For more descriptions including the MVAPICH User Guide and other MVAPICH publications, see http://mvapich.cse.ohio-state.edu.

With Intel-MPI

DRAFT

This article is being reviewed for completeness and technical accuracy.

Intel's MPI library is another alternative for building and running your MPI application. The available Intel MPI modules are:

```
mpi-intel/3.1.038
mpi-intel/3.1b
mpi-intel/3.2.011
```

To use Intel MPI, first create a file $HOME/.mpd.conf that has the single line:

```
MPD_SECRETWORD=sometext
```

('sometext' should be unique for each user)

and change the permission of the file to read/write by you only.

```
%chmod 600 $HOME/.mpd.conf
```

Building Applications

To compile, load an Intel compiler module and an Intel MPI module. Make sure that no other MPI module is loaded (i.e., MPT, MVAPICH or MVAPICH2)

```
%module load mpi-intel/3.1.038
%module load comp-intel/11.1.072
```

Use the **mpiifort/mpiicc** scripts which invoke the Intel ifort/icc compilers.

```
%mpiifort -o your_executable program.f
```

Running Applications

To run it, in your PBS script make sure the intel MPI modules are loaded as above, start the MPD daemon, use mpiexec, and terminate the daemon at the end. For example,

```
#PBS ..
..
module load mpi-intel/3.1.038
module load comp/intel/10.1.021_64

# Note: The following three lines should really be in one line
```

```
mpdboot --file=$PBS_NODEFILE --ncpus=1 --totalnum=`cat $PBS_NODEFILE  |
sort -u | wc -l` --ifhn=`head -1 $PBS_NODEFILE`
 --rsh=ssh --mpd=`which mpd` --ordered

# CPUS_PER_NODE and TOTAL_CPUS below represent numerical numbers
# for the job at hand

mpiexec -ppn CPUS_PER_NODE -np TOTAL_CPUS ./your_executable

# terminate the MPD daemon

mpdallexit
```

With OpenMP

DRAFT

This article is being reviewed for completeness and technical accuracy.

Building Applications

To build an OpenMP application, you need to use the *-openmp* Intel compiler flag:

```
%module load comp-intel/11.1.072
%ifort -o your_executable -openmp program.f
```

Running Applications

The maximum number of OpenMP threads an application can use on a Pleiades node depends on (i) the number of physical processor cores in the node and (ii) if hyperthreading is available and enabled. Hyperthreading technology is not available for the Harpertown processor type. It is available and enabled at NAS for the Nehalem-EP and Westmere-EP processor types. With hyperthreading, the OS views each physical core as two logical processors and can assign two threads to it. This is beneficial only when one thread does not keep the functional units in the core busy all the time and can share the resources in the core with another thread. Running in this mode may take less than 2 times the walltime compared to running only 1 thread on the core.

```
Before running with hyperthreading for your production runs, it is
recommended that you experiment with it to see if it is beneficial
for your application.
```

Maximum Threads

Processor Type	Maximum Threads without Hyperthreading	Maximum Threads with Hyperthreading
Harpertown	8	N/A
Nehalem-EP	8	16
Westmere-EP	12	24

Here is sample PBS script for running OpenMP applications on a Pleiades Nehalem-EP node without hyperthreading:

```
#PBS -lselect=1:ncpus=8:ompthreads=8:model=neh,walltime=1:00:00

module load comp-intel/11.1.072

cd $PBS_O_WORKDIR

./your_executable
```

Here is sample PBS script with hyperthreading:

```
#PBS -lselect=1:ncpus=8:ompthreads=16:model=neh,walltime=1:00:00

module load comp-intel/11.1.072

cd $PBS_O_WORKDIR

./your_executable
```

With SGI's MPI and Intel OpenMP

DRAFT

This article is being reviewed for completeness and technical accuracy.

Building Applications

To build an MPI/OpenMP hybrid executable using SGI's MPT and Intel's OpenMP libraries, your code needs to be compiled with the *-openmp* flag and linked with the *-mpi* flag.

```
%module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789
%ifort -o your_executable prog.f -openmp -lmpi
```

Running Applications

Here is a sample PBS script for running MPI/OpenMP application on Pleiades using 3 nodes and on each node, 4 MPI processes with 2 OpenMP threads per MPI process.

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:model=neh
#PBS -lwalltime=1:00:00

module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789
setenv OMP_NUM_THREADS 2

cd $PBS_O_WORKDIR

mpiexec ./your_executable
```

You can specify the number of threads, *ompthreads*, on the PBS resource request line, which will cause the PBS prologue to set the OMP_NUM_THREADS environment variable.

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:ompthreads=2:model=neh
#PBS -lwalltime=1:00:00

module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789

cd $PBS_O_WORKDIR

mpiexec ./your_executable
```

Performance Issues

For pure MPI codes built with SGI's MPT library, performance on Nehalem-EP and Westmere-EP nodes improves by pinning the processes through setting MPI_DSM_DISTRIBUTE envrionment variables to 1 (or true). However, for MPI/OpenMP codes, all the OpenMP threads for the same MPI process have the same process ID and setting this variable to 1 causes all OpenMP threads to be pinned on the same core and the performance suffers.

It is recommended that MPI_DSM_DISTRIBUTE is set to 0 and *omplace* is to be used for pinning instead.

If you use Intel version 10.1.015 or later, you should also set KMP_AFFINITY to *disabled* or OMPLACE_AFFINITY_COMPAT to *ON* as Intel's thread affinity interface would interfere with dplace and omplace.

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:ompthreads=2:model=neh
#PBS -lwalltime=1:00:00

module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789

setenv MPI_DSM_DISTRIBUTE 0
setnev KMP_AFFINITY disabled

cd $PBS_O_WORKDIR

mpiexec -np 4 omplace ./your_executable
```

With MVAPICH and Intel OpenMP

DRAFT

This article is being reviewed for completeness and technical accuracy.

Building Applications

To build an MPI/OpenMP hybrid executable using MVAPICH and Intel's OpenMP libraries, use *mpif90, mpicc, mpicxx* with the *-openmp* flag.

```
%module load comp-intel/11.1.072  mpi-mvapich2/1.4.1/intel
%mpif90 -o your_executable prog.f90 -openmp
```

Running Applications

With MVAPICH, a user's environment variables (such as VIADEV_USE_AFFINITY and OMP_NUM_THREADS) are not passed in to mpiexec, thus they need to be passed in explicitly, such as with /usr/bin/env.

Here is an example on how to run a MVAPICH/OpenMP hybrid code with a total of 12 MPI processes and 2 OpenMP threads per MPI process:

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:model=neh

module load comp-intel/11.1.072 mpi-mvapich2/1.4.1/intel

mpiexec /usr/bin/env VIADEV_USE_AFFINITY=0 OMP_NUM_THREADS=2 ./your_executable
```

Performance Issues

Setting the environment variable VIADEV_USE_AFFINITY to 0 disables CPU affinity because MVAPICH does its own pinning. Setting it to 1 actually causes multiple OpenMP threads to be placed on a single processor.

Porting to Columbia

Default or Recommended compiler version and options

DRAFT

This article is being reviewed for completeness and technical accuracy.

Intel compiler versions 10.0, 10.1, 11.0 and 11.1 are available on Columbia as modules. Use the 'module avail' command to find available versions.

The current default compiler module on Columbia is *intel-comp.10.1.013*.

In addition to the few flags mentioned in the article Recommended Intel Compiler Debugging Options, here are a few more to keep in mind:

Turn on optimization: *-O3*

If you do not specify an optimization level (*-On*, *n=0,1,2,3*), the default is *-O2*. If you want more aggressive optimizations, you can use *-O3*. Note that using *-O3* may not improve performance for some programs.

Turn inlining on: *-ip* or *-ipo*

Use of *-ip* enables additional interprocedural optimizations for single file compilation. One of these optimizations enables the compiler to perform inline function expansion for calls to functions defined within the current source file.

Use of *-ipo* enables multifile interprocedural (IP) optimizations (between files). When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

Parallelize your code: *-openmp* or *-parallel*

-openmp handles OMP directives and *-parallel* looks for loops to parallelize.

For more compiler/linker options, read **man ifort, man icc**, or

```
%ifort -help
%icc -help
```

Porting to Columbia: With SGI's MPT

DRAFT

This article is being reviewed for completeness and technical accuracy.

The available SGI MPT modules on Columbia are:

```
mpt.1.16.0.0
mpt.1.18.0.0
mpt.1.19.0.0
mpt.1.22.0.0
mpt.1.25
```

The current default version is *mpt.1.16.0.0*.

Environment Variables

On Columbia, when you load any of the above MPT modules, several environment variables such as CPATH, INCLUDE, LD_LIBRARY_PATH, etc., are modified by pre-pending the appropriate MPT directories. Also, the following MPT-related environment variables are modified from their default values for improved performance:

```
setenv MPI_BUFS_PER_HOST 256
setenv MPI_BUFS_PER_PROC 256
setenv MPI_DSM_DISTRIBUTE
```

The meanings of these variables and their default values are:

- MPI_BUFS_PER_HOST

  Determines the number of shared message buffers (16 KB each) that MPI is to allocate for each host (i.e., C21, C22, C23, C24). These buffers are used to send and receive long inter-host messages.

  Default: 32 pages (1 page = 16KB) for mpt.1.16, mpt.1.18, mpt.1.19, mpt.1.22
  Default: 96 pages (a page = 16KB) for mpt.1.25

- MPI_BUFS_PER_PROC

  Determines the number of private message buffers (16 KB each) that MPI is to allocate for each process. These buffers are used to send long messages and intra-host messages.

  Default: 32 pages (1 page = 16KB)

- MPI_DSM_DISTRIBUTE (toggle)

  Activates NUMA job placement mode. This mode ensures that each MPI process gets a unique CPU and physical memory on the host with which that CPU is associated. This feature can also be overridden by using dplace or omplace. This feature is most useful if running on a dedicated system or running within a cpuset.

  Default: Not enabled

Building Applications

Building MPI applications with SGI's MPT library simply requires linking with -lmpi and/or -lmpi++. See the article SGI MPT for some examples.

Running Applications

```
MPI executables built with SGI's MPT are not allowed to run on the
Columbia front-end node.
```

You can run your MPI job on C21 - C24 in an interactive PBS session or through a PBS batch job. Use **mpiexec** (under /PBS/bin) or mpirun to start your MPI processes. For example:

```
#PBS -lncpus=8
....
mpiexec -np N ./your_executable
```

The -np flag (with *N* MPI processes) can be omitted if the value of *N* is the same as the value specified for *ncpus*.

Porting to Columbia: With OpenMP

DRAFT

This article is being reviewed for completeness and technical accuracy.

Building Applications

To build an OpenMP application, you need to use the -openmp Intel compiler flag:

```
%ifort -o your_executable -openmp program.f
```

Note that if you are compiling separate files, then -openmp is required at the link step to link in the OpenMP library.

Running Applications

Note that *OMP_NUM_THREADS* is set to 1 by default for PBS jobs. Reset it to the number of threads that you want.

Here is a sample PBS script for running OpenMP applications on Columbia:

```
#PBS -lncpus=8,walltime=1:00:00

setenv OMP_NUM_THREADS 8

cd $PBS_O_WORKDIR

./your_executable
```

Porting to Columbia: With MPI and OpenMP

DRAFT

This article is being reviewed for completeness and technical accuracy.

Building Applications

To build a hybrid MPI+OpenMP application, you need to compile your code with the -openmp compiler flag and link in both the Intel OpenMP and the SGI MPT library:

```
%ifort -o your_executable -openmp program.f -lmpi
```

Running Applications

Process/thread placement is critical to the performance of MPI+OpenMP hybrid codes. Two environment variables should be set to get the proper placement:

- MPI_DSM_DISTRIBUTE

  Activates NUMA job placement mode. This mode ensures that each MPI process gets a unique CPU and physical memory on the node with which that CPU is associated. Currently, the CPUs are chosen by simply starting at relative CPU 0 and incrementing until all MPI processes have been forked.

- MPI_OPENMP_INTEROP

  Setting this variable modifies the placement of MPI processes to better accommodate the OpenMP threads associated with each process. For this variable to take effect, you must also set MPI_DSM_DISTRIBUTE.

Also note that *OMP_NUM_THREADS* is set to 1 by default for PBS jobs. Reset it to the number of threads that you want.

Here is a sample PBS script for running MPI+OpenMP hybrid (2 MPI processes, 4 OpenMP threads per MPI process) applications on Columbia:

```
#PBS -lncpus=8,walltime=1:00:00

setenv MPI_DSM_DISTRIBUTE
setenv MPI_OPENMP_INTEROP
setenv OMP_NUM_THREADS 4

cd $PBS_O_WORKDIR

mpirun -np 2 ./your_executable
```

Software Environment

Software: Overview

DRAFT

This article is being reviewed for completeness and technical accuracy.

Software on the NAS HECC systems include the operating systems, programming environments, licensed or open source software, etc. The following lists the few directories where you can find most of the software you need.

- */bin* : essential user commands binaries, such as *cp, ls, mv, vi*, etc.
- */lib* : essential shared libraries and kernel modules, such as *libc, libm*, etc.
- */usr/bin* : most user commands, such as *cat, diff, ldd*, etc.
- */usr/lib* : libraries for programming and packages, such as *libstdc++, libGL*, etc.
- */usr/include* : system's general-use include files for the C programming language
- */usr/local/bin* : binaries added for local use, such as *acct_ytd, bbftp*, etc.
- */usr/local/lib* : shell start up files, such as *glocal.cshrc* for NAS systems
- */PBS* : software for submitting, monitoring and managing PBS jobs
- */nasa* : licensed or open source software modules

Except for those under */nasa*, the binaries, libraries and include files above should have been included in your default search path.

Read the articles on Modules to learn how to use licensed or open source software managed by modules.

In addition, on Pleiades there are some useful tools provided by members of the Application Performance and Productivity Group. They are stored under the directory /u/scicon/tools.

Operating Systems

DRAFT

This article is being reviewed for completeness and technical accuracy.

All NAS HECC systems (including Pleiades and Columbia) are running SGI ProPack for Linux which is designed to enhance the Linux experience for SGI systems.

To find the Linux kernel version number on a host, use:

```
%uname -r
```

To find the SGI release number on a host, use:

```
%cat /etc/sgi-release
```

All Pleiades front-ends and compute nodes are running with ProPack 7SP1.

All Columbia systems, including both frond-ends and compute systems, are running with ProPack 6SP5.

Modules

DRAFT

This article is being reviewed for completeness and technical accuracy.

A system called "modules" to centralize the location of licensed products from vendors or software from public domain is installed on all NAS HECC systems.

To use the modules commands, you have to do either one of the following first:

1. Source the following files in your .cshrc or .profile

   in .cshrc (for csh users)

   ```
   source /usr/local/lib/global.cshrc
   ```
   in .profile (for bash users)

   ```
   source /usr/local/lib/global.profile
   ```
2. In the shell that you want to use the module commands, do one of the following:

   (csh users)

   ```
   %source /usr/share/modules/init/csh
   ```
   (bash users)

   ```
   %. /usr/share/modules/init/bash
   ```

The following are useful module commands to remember:

- `%module avail`

  to find out what modules are available.
- `%module list`

  to list which modules are loaded in your environment.
- `%module purge`

  to unload all loaded modulefiles.
- `%module load` *`module_name1 module_name2 ... module_nameN`*

to load the desired modules.

- `%module switch` *`old_module_name new_module_name`*

  to switch between two modules.

- `%module show` *`module_name`*

  to show changes to the environment that will happen if you load *module_name.*

Table of All Modules

DRAFT

This article is being reviewed for completeness and technical accuracy.

The table below shows the available software managed through modules on Pleiades and/or Columbia. To request installation of a software as a module, please send an email to support@nas.nasa.gov

Note that the name of a software module may contain:

- software name
- vendor name
- version number
- varieties such as what compiler and/or what library is used to build the software

For example,

- *comp-intel/11.1.072* represents the Intel Compiler version 11.1.072.
- *mpi-sgi/mpt.2.04.10789* represents the SGI MPI library version mpt.2.04.10789.
- *mpi-mvapich2/1.4.1/intel* represents the MVAPICH2 MPI library version 1.4.1 built with an Intel compiler.

Use the "module avail" command to see all the available versions and provide the full name of a module when you decide to load a module.

Available Modules (as of 30 August 2010)

Software	**Platforms**	**Function**
Intel compiler	Pleiades/Columbia	Compiler
Intel mkl	Pleiades/Columbia	Math/Scientific Library
Intel mpi	Pleiades/Columbia	MPI Library
SGI mpt	Pleiades/Columbia	MPI Library
SGI scsl	Columbia	Math/Scientific Library
automake	Columbia	Makefile Tool
boost	Columbia	C++ Library
cpan	Pleiades	Comprehensive Perl Archive Network
cscope	Columbia	Source Code Browsing Tool
drm	Pleiades	X Window Library Tool
eclipse	Pleiades	Software Development Environment
emacs	Pleiades	Text Editor
ensight	Pleiades/Columbia	Data Visualization and Analysis Tool

fieldview	Pleiades/Columbia	Data Visualization and Analysis Tool
flex	Pleiades	Text Scanner Generation Tool
fluent	Pleiades	CFD Modeling Application
gaussian	Pleiades/Columbia	Quantum Chemistry Application
gcc	Pleiades/Columbia	GNU C/C++ Compiler
gd	Pleiades/Columbia	Images Creation Library
git	Pleiades/Columbia	Version Control System
glib	Pleiades/Columbia	Low-level Core Library
gmp	Pleiades/Columbia	Math Library
gnuplot	Pleiades/Columbia	Data Visualization Tool
grace	Pleiades/Columbia	Data Visualization Tool
grads	Pleiades/Columbia	Data Visualization and Analysis Tool
gridgen	Pleiades/Columbia	CFD Grid Generation Tool
gsl	Pleiades/Columbia	GNU Scientific Library
hcss	Pleiades/Columbia	Herschel Common Science System
hdf4	Pleiades/Columbia	I/O Library and Tools
hdf5	Pleiades/Columbia	I/O Library and Tools
idl	Pleiades/Columbia	Data Visualization and Analysis Tool
idn	Pleiades	GNU Libidn
imagemagick	Pleiades/Columbia	Image Tool
java-sdk	Columbia	Programming Language
jpeg	Columbia	Image Tool
jvm	Pleiades	Java Virtual Machine
libxml	Columbia	C Parser and Toolkit
lsdyna3d	Pleiades/Columbia	Finite Element Application
matlab	Pleiades/Columbia	Numerical Computing Environment and Programming Language
mlp	Columbia	Multi-Level Parallelism Library
mpfr	Pleiades	Multiple-Precision Floating-point Computations Library
mpich2	Columbia	MPI Library
mvapich2	Pleiades	MPI Library
ncarg	Pleiades/Columbia	Graphics Library for Scientifc Data
ncl	Pleiades/Columbia	NCAR Command Language
nco	Pleiades/Columbia	netCDF Operators
netcdf	Pleiades/Columbia	I/O Library
octave	Pleiades/Columbia	Numerical Computations Language
paraview	Pleiades	Data VIsualization and Analysis Tool
parmetis	Pleiades/Columbia	Math/Numerical Library
pdf	Columbia	PDF File Generation Library

perl	Columbia	Programming Language
petsc	Columbia	Math/Numerical Library
parallel netcdf	Pleiades/Columbia	Parallel I/O Library
png	Columbia	Portable Network Graphics Format
pyMPI	Columbia	MPI Program Development with Python
python	Pleiades/Columbia	Programming Language
ruby	Pleiades	Programming Language
svn	Pleiades/Columbia	Revision Control Application
swig	Pleiades/Columbia	Software Development Tool
tcl-tk	Pleiades/Columbia	Scripting Language
tecplot	Pleiades/Columbia	Data Visualization and Analysis Tool
texlive	Pleiades	TeX System Application
totalview	Pleiades/Columbia	Debugger
udunits	Pleiades/Columbia	Data Format Library
visit	Pleiades/Columbia	Data Visualization and Analysis Tool
xv	Pleiades	Images Display Application
xxdiff	Pleiades	Graphical File And Directories Comparator And Merge Tool
yaml	Pleiades/Columbia	Human-Readable Data Serialization Format
zlib	Columbia	Data Compression Library

Licensed Application Software

Licensed Application Software: Overview

DRAFT

This article is being reviewed for completeness and technical accuracy.

A few licensed applications from different vendors are installed on NAS HECC systems under the */nasa* directory. They are either purchased by NAS (with justification that many users need it) or by users themselves. If you would like to use a licensed application which is not yet available on NAS HECC systems, you may have to purchase the license yourself.

Tecplot

DRAFT

This article is being reviewed for completeness and technical accuracy.

Tecplot 360 is a CFD and Numerical Simulation Visualization Software used in post-processing simulation results. Common tasks associated with post-processing analysis of flow solver (e.g. Fluent, STAR-CD, OpenFOAM) can include such tasks as:

- Calculating grid quantities (e.g. aspect ratios, skewness, orthogonality and stretch factors)
- Normalizing data; Deriving flow field functions like pressure coefficient or vorticity magnitude
- Verifying solution convergence
- Estimating the order of accuracy of solutions
- Interactively exploring data through cut planes (a slice through a region), iso-surfaces (3-D maps of concentrations), particle paths (dropping an object in the "fluid" and watching where it goes).

As of Dec. 2008, the Tecplot license at NAS no longer has restrictions on the number of copies of Tecplot that can be run concurrently.

Note: If you have set the stacksize with a command like "limit stacksize unlimited", you will have to reduce the stacksize for Tecplot to run. For example,

```
%limit stacksize 2000000
```

For more information, please visit Tecplot's documentation page.

See also:

http://en.wikipedia.org/wiki/Tecplot

IDL

DRAFT

This article is being reviewed for completeness and technical accuracy.

IDL is a software for data analysis, visualization, and cross-platform application development. IDL combines tools for any type of project, from "quick-look," interactive analysis and display to large-scale commercial programming projects.

For more information, please visit the IDL home page.

There are 6 licenses available for 6 users to use IDL at the same time. If you are not able to use idl because the licenses are being used, try using it at a later time, or issue the command 'lmstat -a' to find out how many licenses are in use.

See also:

http://en.wikipedia.org/wiki/IDL_(programming_language)

LS-DYNA

DRAFT

This article is being reviewed for completeness and technical accuracy.

LS-DYNA is a general-purpose transient dynamic finite element program capable of simulating complex real world problems. It is optimized for shared- and distributed-memory Unix, Linux, and Windows based, platforms.

Current license (good until Aug. 31, 2011) allows upto 4 CPUs.

Typical usage:

```
ls971d NCPUS=$OMP_NUM_THREADS I=**.key
```

```
mpiexec -np xx mpp971d I=**.key
```

Use the lstc_qrun command to check how many CPUs are using the license. Use the lstc_qkill command to release the license if it is not released automatically after a job is terminated.

For more information, please visit the LS-DYNA web page.

See also:

http://en.wikipedia.org/wiki/LS-DYNA

Matlab

DRAFT

This article is being reviewed for completeness and technical accuracy.

Matlab is a numerical computing environment and programming language. Created by The MathWorks, Matlab allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. Although it specializes in numerical computing, an optional toolbox interfaces with the Maple symbolic engine, allowing it to be part of a full computer algebra system.

For more information, please visit the Matlab web site at MathWorks.

```
Note: Matlab 2010 does not work on Pleiades or Columbia yet because
of technical issues.
```

See also:

http://en.wikipedia.org/wiki/Matlab

Gaussian

DRAFT

This article is being reviewed for completeness and technical accuracy.

Gaussian 03 is a suite of electronic structure programs. It is used by chemists, chemical engineers, biochemists, physicists and others for research in established and emerging areas of chemical interest.

Starting from the basic laws of quantum mechanics, Gaussian predicts the energies, molecular structures, and vibrational frequencies of molecular systems, along with numerous molecular properties derived from these basic computation types. It can be used to study molecules and reactions under a wide range of conditions, including both stable species and compounds which are difficult or impossible to observe experimentally such as short-lived intermediates and transition structures.

For more information, please see the Gaussian manual or the Gaussian web site.

Two versions (c.02 and e.01) of Gaussian03 have been installed on Columbia systems. To use the older c.02 version, do the following in your PBS script:

```
module load gaussian.03.c02
source $g03root/g03/bsd/g03.login

g03 input output
```

To use the newer e.01 version (built with intel-comp.10.0.023 and intel-mkl.9.1.023), do:

```
module load gaussian.03.e.01
source $g03root/g03/bsd/g03.login

g03 input output
```

If you are a bash user, then do:

```
. /usr/share/modules/init/bash
module load gaussian.03.e.01
. $g03root/g03/bsd/g03.profile

g03 input output
```

See also:

http://en.wikipedia.org/wiki/GAUSSIAN

FieldView

DRAFT

This article is being reviewed for completeness and technical accuracy.

FieldView is Intelligent Light's CFD post-processing software to quickly identify important flow features and characteristics in simulations. It allows interactive exploration for thorough understanding of results. You can use it to examine and compare cases, extract critical values, and make presentations.

Current license allows up to 4 concurrent uses.

For more information, see Intelligent Light's FieldView home page.

Ensight

DRAFT

This article is being reviewed for completeness and technical accuracy.

EnSight is a software package from CEI that is used for analyzing, visualizing and communicating high-end scientific and engineering datasets. It is a post processing environment with an extensive list of features.

Please see the CEI EnSight home page to get more information.

Gridgen

DRAFT

This article is being reviewed for completeness and technical accuracy.

Gridgen is Pointwise's meshing software used by engineers and scientists to generate high quality grids for engineering analysis.

For more information, please visit the Gridgen home page at the Pointwise web site.

Running Jobs with PBS

Portable Batch System (PBS): Overview

All NAS facility supercomputers use the Portable Batch System (PBS) from Altair for batch job submission, job monitoring, and job management. Note that different systems may use different versions of PBS, so the available features may vary slightly from system to system.

Batch Jobs

Batch jobs run on compute nodes, not the front-end nodes. A PBS scheduler allocates blocks of compute nodes to jobs to provide exclusive access. You will submit batch jobs to run on one or more compute nodes using the *qsub* command from an interactive session on one of the front-end nodes (such as, pfe[1-12], bridge[1-2] for Pleiades or cfe2 for Columbia).

Normal batch jobs are typically run by submitting a script. A "jobid" is assigned after submission. When the resources you request become available, your job will execute on the compute nodes. When the job is complete, the PBS standard output and standard error of the job will be returned in files available to you.

Take carefully note when porting job submission scripts from systems outside of the NAS environment or between the Pleiades and Columbia supercomputers you may need to make changes to your existing scripts to make them work properly on these systems.

Interactive Batch Mode

PBS also supports an interactive batch mode, using the *qsub -I*, where the input and output is connected to the user's terminal, but the scheduling of the job is still under control of the batch system.

Queues

The available queues on different systems vary, but all typically have constraints on maximum wall-time and/or the number of CPUs allowed for a job. Some queues may also have other constraints or be restricted to serving certain users or groups. In addition, to ensure that each NASA mission directorate is granted their allocated share of resources at any given time, mission directorate limits (called "shares") are also set on Pleiades and Columbia.

See **man pbs** for more information.

Job Accounting

DRAFT

This article is being reviewed for completeness and technical accuracy.

Usage on the HECC machines at NAS, except for the front-end machines, is charged.

Starting May 1, 2011, the accounting unit is the Standard Billing Unit (SBU). The SBUs charged to a PBS job running on the compute node(s) is:

*SBU charged = Wall_Clock_Hours_Used * Number of MAUs * SBU Rate*

where the MAU represents the minimum allocatable unit of resources available through PBS. On Pleiades, an MAU is a node (with 8 cores for Harpertown and Nehalem-EP or 12 cores for Westmere in each node). On Columiba, an MAU has 4 cores. Charging is based on the number of MAUs allocated to a job, not how many cores are actually used during run-time. Once a user is allocated the resources, that user has exclusive access to those resources until the user's job completes or exceeds its requested wall-clock time.

The SBU rate for each of the NAS processors is outlined below:

```
Host                            SBU Rate (per MAU)
Pleiades Westmere-EP nodes            1.00
Pleiades Nehalem-EP nodes             0.80
Pleiades Harpertown nodes             0.45
Columbia Itanium-2                    0.18
```

In addition, charges on Columbia apply both to jobs that run successfully and those that are interrupted. Interrupted jobs are charged by taking the elapsed job time in hours, subtracting 1 hour, multiplying that by the number of MAUs used, and then deducting the resulting amount from the allocation. (Users are encouraged to have their applications checkpoint roughly every hour.)

In the near future, interruptions on Pleiades will be handled in a similar manner.

For example:

If you have a 24-hour job on Columbia that requires 16 MAUs (i.e., 64 cores), It has run for 12 hours and the system crashes. The accounting system will take the 12 hours, subtract 1 hour, and compute the SBUs (11 hours X 16 MAUs x 0.18 = 31.68 SBUs), which will then be subtracted from your allocation for your GID.

Job Accounting Utilities

DRAFT

This article is being reviewed for completeness and technical accuracy.

The job accounting utilities "acct_ytd" and "acct_query" can be used to obtain resource usage and charging information about your account, the accounts of other users on your project, and the project as a whole. Daily usage totals for each account are available for the current operational period.

- **acct_ytd**

  The "acct_ytd" command provides a year-to-date summary of accounting information for groups to which a user belongs. It will normally be accurate as of midnight the previous night, when accounting was last run.

  A number of parameters can be used with "acct_ytd", but the simpliest way is to type "acct_ytd" on a host without any parameters. This produces a line of output for each project you have access to on that host.

  ```
  %acct_ytd
  ```
  You can also specify the host group and/or a specific GID (for example, a0800).

  ```
  %acct_ytd -cpleiades a0800   %acct_ytd -ccolumbia a0800
  ```

  To find the allocations and usages of all your GIDs on all hosts, use the -call flag.

  ```
  %acct_ytd -call
  ```
  See **man acct_ytd** on Pleiades and Columbia for more information.
- **acct_query**

  The "acct_query" command searches and displays process-level billing records. This means that while totals over a period or for each day in a period are possible, you can also obtain detailed billing records for each process run in a period.

  For example, to see all the SBU usage, beginning June 1, 2010, ending July 1, 2010, for all projects and on all hosts by user zsmith:

  ```
  %acct_query -b06/01/10 -e07/01/10 -pall -call -uzsmith
  ```
  To see the current SBU usage for the operational year 2010 (defined as May 1, 2010 to May 1, 2011 for most mission directorates) for all projects and on all hosts by user zsmith:

  ```
  %acct_query -y10 -pall -call -uzsmith
  ```

Eligible hostnames include:

1. columbia21
2. columbia22
3. columbia23
4. columbia24
5. pbs1
6. pleiades (for Harpertown nodes)
7. pleiades_N (for Nehalem nodes)
8. pleiades_W (for Westmere nodes)

See **man acct_query** on Pleiades and Columbia for more information.

Multiple GIDs and Charging to a specific GID

DRAFT

This article is being reviewed for completeness and technical accuracy.

Each approved project is assigned a project id (GID). Members of a GID are authorized to use the resources allocated to that GID. For those users who have access to multiple GIDs, be aware that only one of those GIDs is considered your default.

Use the "groups" command to find which GIDs you are a member of. The following example shows that user zsmith is a member of the groups a0800, a0907, all, and e0720.

```
%groups zsmith
zsmith : a0800 a0907 all e0720
```

The first GID from the "groups" list should be your default GID. This can be verified through the /etc/passwd file. For example, the /etc/passwd file has an entry for user zsmith with the GID 20800 (which is the same as a0800, his default GID).

```
%grep zsmith /etc/passwd
zsmith:x:6666:20800:Z. Smith,,650-604-4444,:/u/zsmith:/bin/csh
```

When you use resources on the compute nodes through PBS jobs, SBUs are deducted from your default GID unless you specify otherwise. To charge resource usage to an alternative GID for a batch job, you can use the PBS flag "-W group_list=*account*" either in your script or on the "qsub" command line. For example:

```
#PBS -W group_list=a0907
```
or

```
%qsub -W group_list=a0907
```

Commonly Used PBS Commands

DRAFT

This article is being reviewed for completeness and technical accuracy.

man pbs provides a list of all PBS commands. The four most commonly used PBS commands, qsub, qstat, qdel and qhold, are briefly described below.

- **qsub**
    - Submit a batch job to the specified queue using a script

      ```
      %qsub -q queue_name job_script
      ```
      Common possibilities for *queue_name* at NAS include *normal, debug, long,* and *low*. When *queue_name* is omitted, the job is routed to the default queue, which is the *normal* queue.
    - Submit an interactive PBS job

      ```
      %qsub -I -q queue_name -lresource_list

      No job_script should be included when submitting an
      interactive PBS job.
      ```
      The *resource_list* typically specifies the number of nodes, cpus, amount of memory and walltime needed for this job. The following example shows a request for Pleides with 2 nodes, 8 cpus per node, and a walltime limit of 3 hours.

      ```
      %qsub -I -lselect=2:ncpus=8,walltime=3:00:00
      ```
      See **man pbs_resources** for more information on what resources can be specified. If -l*resource_list* is omitted, the default resources for the specified queue is used. When *queue_name* is omitted, the job is routed to the default queue, which is the *normal* queue.
- **qstat**
    - Display queue information

      ```
      %qstat -Q queue_name
      %qstat -q queue_name
      $qstat -fQ queue_name
      ```
      These commands display in different formats all the queue available on the systems, their constraints and status. The *queue_name* is optional.
    - Display job status

◊ Display all jobs in any status (running, queued, held)
```
%qstat -a
```

◊ Display all running or suspended jobs
```
%qstat -r
```

◊ Display the execution hosts of the running jobs
```
%qstat -n
```

◊ Display all queued, held or waiting jobs
```
%qstat -i
```

◊ Display jobs that belong to the specified user
```
%qstat -u user_name
```

◊ Display any comment added by the administrator or scheduler
```
%qstat -s
```
This option is typically used to find clues of why a job has not started running.

◊ Display detailed information about a specific job
```
%qstat -f job_id
```

◊ Display status informaton for finished jobs (within the past 7 days)
```
%qstat -xf job_id
%qstat -xu user_id
```

This option is only available in newer version of PBS, which has been installed on Pleiades, but not on Columbia.

`Some of these flags can be combined when checking the job status.`

- **qdel**

  Delete a job

  ```
  %qdel job_id
  ```
- **qhold**

  Hold a job

  ```
  %qhold job_id
  ```
  Only the job owner or a system administrator with su or root privilege can place a hold on a job. The hold can be released using the "qrls" command.

For more detailed information on each command, see their corresponding **man pages**.

The *devel* queue on Pleiades is served by a non-default PBS server, pbspl3, and the syntax for qsub, qstat, and qdel jobs in the *devel* queue needs to include pbspl3. Read this article for more information.

Commonly Used QSUB Options in PBS Scripts or in the QSUB Command Line

DRAFT

This article is being reviewed for completeness and technical accuracy.

The "qsub" options can be read from the PBS directives of a PBS *job_script* or from the qsub command line. For a complete list of available options, see **man qsub**. The more commonly used ones are listed below.

-S *shell_name*
: Specifies the shell that interprets the job script

-V
: Declares that all environment variables in the qsub command's environment are to be exported to the batch job

-v *variable_list*
: Lists environment variables to be exported to the job

-q *queue_name*
: Defines the destination of the job. The common possibilities for *queue_name* on Pleides and Columbia include *normal, debug, long,* and *low*

: The *devel* queue on Pleiades is served by a non-default PBS server, pbspl3, and the syntax for qsub jobs to the *devel* queue needs to include pbspl3. Read this article for more information.

-l *resouce_list*
: Specifies the resources that are required by the job and establishes a limit to the amount of resources that can be consumed. Commonly used resource items are slect, ncpus, walltime, and memory. See **man pbs_resources** for a complete list of available resources.

-e *path*
: Directs the standard error output produced by the request to the stated file path

-o *path*
: Directs the standard output produced by the request to the stated file path.

-j *join*
: Declares that the standard output and error streams of the job should be merged (joined). The values for *join* can be:

oe standard output and error streams are merged in the standard output file
eo standard error and output streams are merged in the standard error file

-m *mail_options*
Defines the set of conditions under which the execution server will send mail message about the job. See **man qsub** for a list of mail_options.

-N *name*
Declares a name for the job

-W *addl_attributes*
Allows for the specification of additional job attributes
The most common ones are
-W group_list=*g_list* specifies the group the job runs under
-W depend=afterany:*job_ID.server_name.nas.nasa.gov* (for example, 12345.pbspl1.nas.nasa.gov) submits a job which is to be executed after *job_ID* has finished with any exit status
-W depend=afterok:*job_ID.server_name.nas.nasa.gov* (for example, 12345.pbspl1.nas.nasa.gov) submits a job which is to be executed after *job_ID* has finished with no errors

-r *y|n*
Declares whether the job is rerunnable

The top of a PBS *job_script* contains PBS directives, each of which begins with the string "#PBS". Here is an example for use on Pleiades.

```
#PBS -S /bin/csh
#PBS -V
#PBS -q long
#PBS -lselect=2:ncpus=8:mpiprocs=4:model=har,walltime=24:00:00
#PBS -j oe
#PBS -o /nobackup/zsmith/my_pbs_output
#PBS -N my_job_name
#PBS -m e
#PBS -W group_list=a0907
#PBS -r n
```

The resources and/or attributes set using options to the "qsub" command line override those set in the directives in the PBS *job_script*.

New Features in PBS

DRAFT

This article is being reviewed for completeness and technical accuracy.

Some of the new features relevant to users are listed below:

- Estimate job start times (version 10.4)

  PBS can estimate the start time for jobs. To show the estimated start times (in the *Est Start* field), use

  ```
  %qstat -T

  This feature is still under testing by NAS system
  administrators and is not yet available to users.
  ```

- Show the processor model (version 10.4)

  Processor model (for example, Harpertwon, Nehalem-EP, and Westmere-EP) can be displayed with

  ```
  %qstat -W o=+model
  ```

- Show job history (version 10.1)

  Use the PBS "-x" option to obtain job history information, including the submission parameters, start/end time, resources used, etc., for jobs that finished executation, were deleted or are still running.

  The job history for finished jobs is preserved for a specific duration. After the duration has expired, PBS deletes the job history information and it is no longer available. Currently, the duration is set to be **7 days** on Pleiades.

  ```
  %qstat -fx job_id
  ```

- Advance and Standing reservations (version 9.2)

  An advance reservation can be made for a set of resources for a specified time. The reservation is only available to a specific user or group of users.

  A standing reservation is an advance reservation which recurs at specified times. For example, the user can reserve 8 nodes every Wednesday from 5pm to 8pm, for the next month.

The reservation is made using the "pbs_rsub" command. PBS either confirms that the reservation can be made, or rejects the request. Once he reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

The following example shows the creation of an advance reservation asking for 1 node with 8 cpus, a start time of 11:30 and a duration of 30 minutes.

```
%pbs_rsub -R 1130 -D 00:30:00 -l select=1:ncpus=8
```

A reservation can be deleted using the "pbs_rdel" command.

For more information, see **man pbs_rsub** and **man pbs_rdel**.

```
Requests to use advance and standing reservations must be
approved by NAS management. Only staff with special privilege
can create the reservations for users.
```

Checkpointing and Restart

DRAFT

This article is being reviewed for completeness and technical accuracy.

None of the NAS HEC systems has an automatic checkpoint capability made available by the operating system. For jobs that need lots of resources and/or long wall-time, you should have a checkpoint/restart capability implemented in the source code or job script.

PBS automatically restarts unfinished jobs after system crashes. If you do not want PBS to restart your job, make sure to add the following in your PBS script:

```
#PBS -r n
```

PBS Environment Variables

DRAFT

This article is being reviewed for completeness and technical accuracy.

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters "PBS_". Some are then followed by a capital O ("PBS_O_") indicating that the variable is from the job's originating environment (i.e. the user's).

The following lists a few useful PBS environment variables.

PBS_O_WORKDIR
: contains the name of the directory from which the user submitted the PBS job

PBS_O_PATH
: value of **PATH** from submission environment

PBS_JOBID
: contains the PBS job identifier

PBS_JOBDIR
: pathname of job-specific staging and execution directory

PBS_NODEFILE
: contains a list of vnodes assigned to the job

TMPDIR
: The job-specific temporary directory for this job
defaults to /tmp/pbs.*job_id* on the vnodes

PBS Scheduling Policy

DRAFT

This article is being reviewed for completeness and technical accuracy.

This article gives a simplified explanation of the PBS scheduling policy on Pleiades and Columbia

PBS scheduling policies change frequently, in response to varying demands and workloads. The current policy (March 1, 2011), simplified, states that jobs are sorted in the following order: current mission directorate CPU use, job priority, queue priority, and job size (wide jobs first).

In each scheduling cycle, PBS examines the jobs in sorted order, starting a job if it can. If the job cannot be started immediately, it is either scheduled around or simply bypassed for this cycle.

There are numerous reasons why jobs won't start, such as:

- The queue is at its running job limit
- You are at your running job limit
- The queue is at its CPU limit
- The mission directorate is at its CPU share limit and the job cannot borrow from another mission
- Not enough CPUs are available

Notice that a high-priority job might be blocked by some limit, while a lower priority job, from a different user or asking for fewer resources, might not be blocked.

If your job is waiting in the queue, use the following commands to get some information about why it has not started running.

pfe1% qstat -s *jobid*
or
pfe1% qstat -f *jobid* | grep -i comment

On Pleiades, output from the following command shows the amount of resources (broken down into Harpertown, Nehalem, and Westmere processors) used and borrowed by each mission directorate, and the resources each mission is waiting for:

pfe1% /u/scicon/tools/bin/qs

The following command provides the order of jobs that PBS schedules to start at the current scheduling cycle. It also provides information regarding processor type(s), mission, and job priority:

```
pfe1% qstat -W o=+model,mission,pri -i
```

The policy described above could result in a large, high-priority job being blocked forever by a steady stream of smaller, low-priority jobs. To prevent jobs from languishing in the queues for an indefinite time, PBS reserves resources for the top N jobs (currently, N is 4), and doesn't allow lower priority jobs start if they would delay the start time of one of the top job ("backfilling"). Additional details are given below.

PBS Sorting Order

- **Mission shares**

  Each NASA mission directorate is allocated a certain percentage of the CPUs in the system. (See Mission Shares Policy on Pleiades .) A job cannot start if that action would cause the mission to exceed its share, unless another mission is using less than its share and has no jobs waiting. In this case, the high-use mission can "borrow" CPUs from the lower-use mission for up to a specified time (currently, max_borrow is 4 hours).

  So , if the job itself needs less than max_borrow hours to run, or if a sufficient number of other jobs from the high-use mission will finish within max_borrow hours to get back under its mission share, then the job can borrow CPUs.

  When jobs are sorted, jobs from missions using less of their share are picked before jobs from missions using more of their share.
- **Job priority**

  Job priority has three components. First is the native priority (the -p parameter to qsub or qalter). Added to that is the queue priority. If the native priority is 0, then a further adjustment is made based on how long the job has been waiting for resources. Waiting jobs get a "boost" of up to 20 priority points, depending on how long they have been waiting and which queue they are in.

  This treatment is modified for queues assigned to the Exploration Systems Mission Directorate (ESMD). For those queues, job priority is set by a separate set of policies controlled by ESMD management.
- **Queue priority**

  Some queues are given higher or lower priorities than the default. (Run "qstat -Q" to get current values.) Note that because the mission share is the most significant sort criterion, job and queue priorities have little effect mission-to-mission.
- **Job size**

Jobs asking for more CPUs are favored over jobs asking for fewer. The reasoning is that, while it is easier for narrow jobs to fill in gaps in the schedule, wide jobs need help collecting enough CPUs to start.

Backfilling

As mentioned above, when PBS cannot start a job immediately, if it is one of the first N such jobs, PBS sets aside resources for the job before examining other jobs. That is, PBS looks at the currently running jobs to see when they will finish (using the wall-time estimates). From those finish times, PBS decides when enough resources (such as CPUs, memory, mission share, and job limits) will become available to run the top job.

PBS then creates a virtual reservation for those resources at that time. Now, when PBS looks at other jobs to see if they can start immediately, it also checks whether starting the job would collide with one of these reservations. Only if there are no collisions will PBS start the lower priority jobs.

This description applies to both Pleiades and Columbia, although the specific queues, priorities, mission percentages, and other elements differ between the two systems.

PBS exit codes

DRAFT

This article is being reviewed for completeness and technical accuracy.

Do we need to have more details for some of the < 0 exit codes? - rh

The PBS exit value of a job may fall in one of four ranges:

- X = 0 (= JOB_EXEC_OK)

  This is a PBS special return value indicating that the job executed successfully
- X < 0

  This is a PBS special return value indicating that the job could not be executed. These negative values are listed below:

  - -1 = JOB_EXEC_FAIL1

    job exec failed, before files, no retry

  - -2 = JOB_EXEC_FAIL2

    job exec failed, after files, no retry

  - -3 = JOB_EXEC_RETRY

    job exec failed, do retry

  - -4 = JOB_EXEC_INITABT

    job aborted on MOM initialization

  - -5 = JOB_EXEC_INITRST

    job aborted on MOM init, checkpoint, no migrate

  - -6 = JOB_EXEC_INITRMG

    job aborted on MOM init, checkpoint, ok migrate

  - -7 = JOB_EXEC_BADRESRT

    job restart failed

- ♦ -8 = JOB_EXEC_GLOBUS_INIT_RETRY

  Init. globus job failed. do retry
- ♦ -9 = JOB_EXEC_GLOBUS_INIT_FAIL

  Init. globus job failed. no retry

- 0 <= X < 128 (or 256 depending on the system)

  This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the .logout script if the user has such a script (csh).
- X >=128 (or 256 depending on the system)

  This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9).

Front-End Usage Guidelines

Pleiades Front-End Usage Guidelines

DRAFT

This article is being reviewed for completeness and technical accuracy.

The front-end systems pfe[1-12] and bridge[1,2] provide an environment that allows you to get quick turnaround while performing the following:

- file editing
- compiling
- short debugging and testing session
- batch job submission to the compute systems

Bridge[1,2], with 4 times the memory on pfe[1-12] and better interconnects, can also be used for the following two functions:

1. **Post processing**

   These nodes have 64-bit versions of IDL, Matlab, and Tecplot installed and have 64 GB of memory (4 times the amount of memory on pfe[1-12]). The bridge nodes will run these applications much faster than on pfe[1-12].

2. **File transfer between Pleiades and Columbia or Lou**

   Note that both the Pleiades Lustre filesystems (/nobackupp[10-70]) and the Columbia CXFS filesystems (/nobackup1[1-h], /nobackup2[a-i]) are mounted on the bridge nodes.

   To copy files between the Pleiades Lustre and Columbia CXFS filesystems, log in to bridge[1,2] and use the *cp* command to perform the transfer. The 10 Gigabit Ethernet (GigE) connections on the two bridge nodes are faster than the 1 GigE used on pfe[1-12], therefore, file transfer out of Pleiades is improved when using the bridge nodes.

   File transfers from bridge[1,2] to Lou[1,2] will go over the 10 GigE interface by default. The commands *scp*, *bbftp*, and *bbscp* are available to do file transfers. Since *bbscp* uses almost the same syntax as *scp*, but performs faster than *scp*, we recommend using *bbscp* over *scp* in cases where you do not require the data to be encrypted when sent over the network.

```
The pfe systems ([pfe1-12]) have a 1 GigE connection, which
can be saturated by a single secure copy (scp). You will see
bad performance whenever more than one file transfer is
happening. Use of bridge1 and bridge2 for file transfers is
strongly recommended.
```

File transfers from the compute nodes to Lou must go through pfe[1-12] or bridge[1,2] first, although going through bridge[1,2] is preferred for performance consideration. See Transferring Files from the Pleiades Compute Nodes to Lou for more information.

When sending data to Lou[1-2], please keep your largest individual file size under 1 TB, as large files will keep all of the tape drives busy, preventing other file restores and backups. To prevent the filesystems on Lou[1-2] from filling up, please limit total data transfers to 1 TB and then wait an hour before continuing. This allows the tape drives to write the data to tape.

Additional restrictions apply to using these front-end systems:

1. No MPI jobs are allowed to run on pfe[1-12], bridge[1,2]
2. A job on pfe[1-12] should not use more than 8 GB. When it does, a courtesy email is sent to the owner of the job.
3. A job on bridge[1,2] should not use more than 56 GB. When it does, a courtesy email is sent to the owner of the job.

Columbia Front-End Usage Guidelines

DRAFT

This article is being reviewed for completeness and technical accuracy.

The front-end system, cfe2, provide an environment that allows users to get quick turnaround while performing the following: file editing; file management; short debugging and testing sessions; and batch job submission to the compute systems.

Running long and/or large (in terms of memory and/or number of processors) debugging or production jobs interactively or in the background of cfe2 is considered to be inconsiderate behavior to the rest of the user community. If you need help submitting such jobs to the batch systems, please contact a NAS scientific consultant at (650) 604-4444 or 1-800-331-USER or send e-mail to: support@nas.nasa.gov

Jobs that cause significant impact on the system load of the Columbia front-end machine (cfe2) are candidates for removal in order to bring the front-end systems back to a normal and smooth environment for all users. A *cron* job regularly monitors the system load and determines if job removal is necessary. The criteria for job removal are described below. Owners of any removed jobs will receive a notification e-mail.

1. To be eligible for removal, the number of processors a front-end interactive job uses can be one (1) or more. Exceptions to this are those programs, utilities, etc. common to users and/or NASA missions that are listed in an "exception file". Examples of these would be:

   bash cp csh emacs gzip rsync scp sftp sh ssh tar tcsh

   Users can submit program names to be added to this exception file by mailing requests to: support@nas.nasa.gov
2. For qualifying processes, the CPU time usage of each process in a job has, on the average, exceeded a threshold defined as:

   (20 min x 8 / number of processes for the job)

   That is, a baseline for removal is a job with 8 processors running for more than 20 minutes. The maximum amount of time allowed for each processor in a job is scaled using the formula:

   20 min x 8 cpu / number-of-processes

   Therefore, the following variations are possible:

   - 160 minutes = (20 * 8) / 1 cpu

- 80 minutes = (20 * 8) / 2 cpu
- 40 minutes = (20 * 8) / 4 cpu
- 20 minutes = (20 * 8) / 8 cpu
- 10 minutes = (20 * 8) / 16 cpu
- 5 minutes = (20 * 8) / 32 cpu
- 2.5 minutes = (20 * 8) / 64 cpu

The conditions of removal are subject to change, when necessary.

PBS on Pleiades

Overview

Overview

On Pleiades, PBS (version 10.4) is used to manage batch jobs that run on the compute nodes (3 different processor types, 9,984 nodes and 91,136 cores in total). PBS features that are common to all NAS systems are described in other articles. Read the following articles for Pleiades-specific PBS information:

- queue structure

- resource request examples
- default variables set by PBS
- sample PBS scripts

Queue Structure

Users should be aware of the PBS queue structure. To find the maximum and default NCPUS (number of CPUs), the maximum and default wall time, the priority of the queue, and whether the queue is disabled or stopped, use the command:

```
%qstat -Q
```

This command also provides statistics of jobs in the states of queued (Q), held (H), running (R), or exiting (E).

Note that the queue structure will change from time to time. Below is a snapshot of the output from this command on June 16, 2011.

```
%qstat -Q
Queue    Ncpus/      Time/         State counts
name       max/def    max/def   jm T/_Q/H/W/__R/E/B pr  Info
======== =====/=== ======/===== == ================ === ========
normal      --/  8  08:00/01:00 -- 0/20/4/0/_60/0/0   0
debug     1025/  8  02:00/00:30 -- 0/_3/0/0/__4/0/0  15
low         --/  8  04:00/00:30 -- 0/_0/0/0/__0/0/0 -10
long      8192/  8 120:00/01:00 -- 0/_8/1/0/206/0/0   0
route       --/  8     --/   -- -- 0/_0/0/0/__0/0/0   0
idle        --/ --     --/   -- -- 0/_0/0/0/__0/0/0   0 disabled
alphatst    --/ -- 120:00/01:00 -- 0/_0/0/0/__0/0/0   0
ded_time    --/ --     --/01:00 -- 0/_0/0/0/__0/0/0   0
devel     4800/  1  02:00/   -- -- 0/_1/0/0/__5/0/0   0
wide        --/ -- 120:00/01:00 -- 0/_1/0/0/__0/0/0  45 disabled
testing     --/ --     --/00:30 -- 0/_0/0/0/__0/0/0   0
somd_spl    --/  8 240:00/01:00 -- 0/_0/0/0/__2/0/0  25
armd_spl  4900/  8 120:00/01:00 10 0/_0/0/0/__0/0/0  15
normal_N    --/  8 120:00/01:00 -- 0/_5/0/0/_10/0/0   0
rtf         --/  8     --/01:00 -- 0/_0/0/0/__0/0/0  65
dpr         --/  8     --/00:10 -- 0/_0/0/0/__0/0/0   0
normal_W    --/  8 120:00/01:00 -- 0/60/5/0/_18/0/0   0
S1848368   744/  1  04:00/   -- -- 0/_0/0/0/__0/0/0   0
kepler      --/  8 120:00/01:00 -- 0/12/0/0/_33/0/0  20
diags       --/ -- 120:00/01:00 -- 0/_0/0/0/__0/0/0   0
```

The *devel* queue on Pleiades is served by pbspl3 (a non-default PBS server). The *devel* queue requires pbspl3 to be included in the syntax for qsub, qstat, and qdel. For more information, read the article Pleiades devel Queue.
To view more information about each queue, use:

```
%qstat -fQ queue_name
```

In the output of this command, you will find additional information such as:

acl_groups
    Lists all GIDs that are allowed to run in the queue.
    For the *normal, debug, long, low* and *wide* queues, all GIDs should be included.
    Special queues, such as *esmd_spl, armd_spl, somd_spl, clv_spl*, etc., typically allow

a few GIDs only.

default_chunk.model

Specifies the default processor model, if you do not specify the processor model yourself.
All queues, except *normal_N* and *normal_W*, default to using nodes with Harpertown model processors.

resources_min.ncpus

If defined, specifies the minimum NCPUs required for the queue.
The *wide* queue requires using a minimum of 1024 CPUs.

max_user_run

If defined, specifies the maximum number of jobs each user is allowed to run in the queue.
For the *normal* queue, it is set at 10. For the *debug* queue, it is set at 2.

`The` *`normal_N`* `and` *`normal_W`* `queues will be removed in the near future. To request using the Nehalem-EP or Westmere nodes, use "model=neh" or "model=wes" attribute in your` *`resource_list`*`. To explicitly request Harpertown nodes, use "model=har". You can apply the model attribute to any queue.`

For example:

```
#PBS -q long
#PBS -l select=1:ncpus=12:model=wes
```

Mission Shares Policy on Pleiades

DRAFT

This article is being reviewed for completeness and technical accuracy.

Mission Directorate shares have been implemented on Pleiades since Feb. 10, 2009. Implementing shares guarantees that each Mission Directorate gets its fair share of resources.

The share to which a job is assigned is based on the GID used by the job. Once all the cores within a Mission Directorate's share have been assigned, other jobs assigned to that share must wait, even if cores are available in a different Mission Directorate's share, with the following exception:

When a Mission Directorate is not using all of its cores, other Mission Directorates can borrow those cores, but only for jobs that will finish within 4 hours. When part of the resource is unavailable, the total number of cores decreases, and each Mission Directorate loses a proportionate number of cores.

You can display the share distribution by adding the "-W shares=-" option to the qstat command. For example:

```
%qstat -W shares=-

Group   Share% Use%  Share Exempt   Use  Avail Borrowed Ratio Waiting
------- ------ ---- ------ ------ ----- ------ -------- ----- -------
Overall    100    0 159748      0   960 158788        0  0.01     960
 ARMD       24   18  38109      0 29680   8429        0  0.78   22512
 HEOMD      23   21  36521      0 34312   2209        0  0.94   28416
 SMD        51   50  80981      0 80968     13        0  1.00  113920
 NAS         2    0   3175      0     0   3175        0  0.00   20240
```

Mission shares are calculated by combining the mission's HECC share of the shared assets combined with the mission-specific assets. The mission shares on Oct 3, 2011 are shown in the second column of the above display. The amount of resources used and borrowed by each mission and resources each mission is waiting for are also displayed.

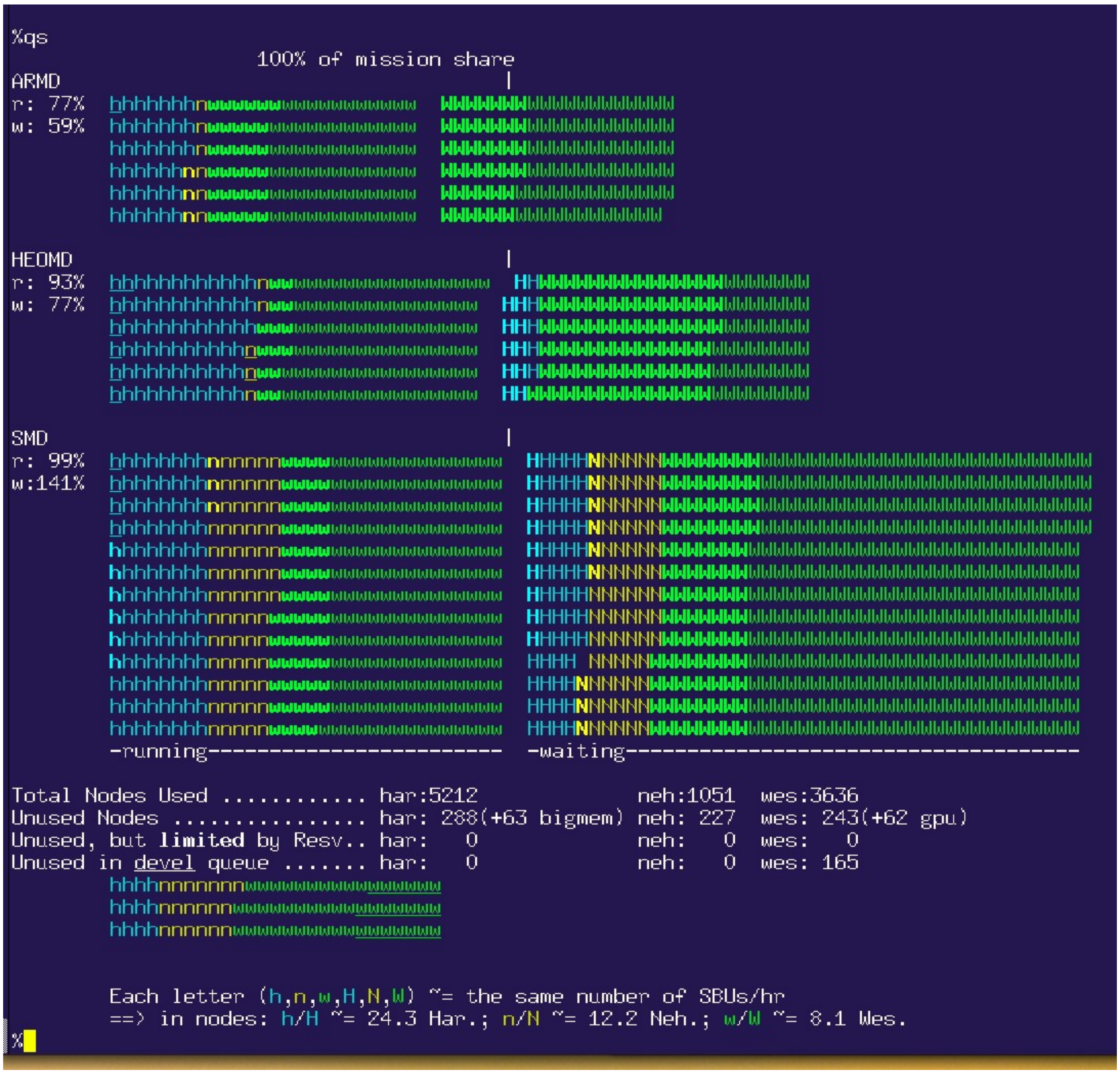
An in-house utility, *qs*, provide similar information with details that break the resources into the Harpertown, Nehalem-EP and Westmere-EP processor types and is available at /u/scicon/tools/bin/qs.

The -h option of qs provides instructions on how to use it:

```
% /u/scicon/tools/bin/qs -h
```

```
usage: qs [-u] [-n N] [-b] [-p] [-d] [-r] [-f M,N] [-q N] [-t] [-v] [-h] [--file f]

          -u       : show used resources only; don't show queued jobs

          -n N     : show time remaining before N nodes are free

          -b       : order segments in bars to help understand borrowing

          -p       : plain output: i.e. no colors or highlights

          -d       : darker colored resource bars (for a light background)

          -r       : use Reverse video for displaying resource bars

          -f M,N   : highlight nodes for jobs that finish in <= M minutes

                     and <= N minutes [default M=60,N=240]

                     (0 turns off highlighting)

          -q N     : highlight nodes for jobs queued in last N minutes [3]

                     (0 turns off highlighting)

          -t       : show time remaining & nodes used for each running job

          --file f : reserved for debugging

          -v       : (verbose) provide explanation of display elements

          -h       : provide this message
```

Here is a sample output file of *qs*:



```
%qs
                    100% of mission share
ARMD                                    |
r: 77%  hhhhhhhnwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
w: 59%  hhhhhhhnwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
        hhhhhhhnwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
        hhhhhhnnwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
        hhhhhhnnwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
        hhhhhhnnwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWW

HEOMD                                   |
r: 93%  hhhhhhhhhhhhnwwwwwwwwwwwwwwwwww   HHWWWWWWWWWWWWWWWWWWWWWW
w: 77%  hhhhhhhhhhhhnwwwwwwwwwwwwwwwww  HHHWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhhhhhwwwwwwwwwwwwwwwwww  HHHWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhhhhnwwwwwwwwwwwwwwwwww  HHHWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhhhhnwwwwwwwwwwwwwwwwww  HHHWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhhhhnwwwwwwwwwwwwwwwwww  HHWWWWWWWWWWWWWWWWWWWWWWW

SMD                                     |
r: 99%  hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
w:141%  hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwww  HHHH NNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwww  HHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwww  HHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwww  HHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
        -running------------------------  -waiting-------------------------------------

Total Nodes Used ............ har:5212             neh:1051  wes:3636
Unused Nodes ................ har: 288(+63 bigmem) neh: 227  wes: 243(+62 gpu)
Unused, but limited by Resv.. har:    0            neh:    0  wes:    0
Unused in devel queue ....... har:    0            neh:    0  wes: 165
        hhhhnnnnnnnwwwwwwwwwwwwwwww
        hhhhnnnnnnwwwwwwwwwwwwwwwww
        hhhhnnnnnnwwwwwwwwwwwwwwwww


        Each letter (h,n,w,H,N,W) ~= the same number of SBUs/hr
        ==> in nodes: h/H ~= 24.3 Har.; n/N ~= 12.2 Neh.; w/W ~= 8.1 Wes.
%
```

Resources Request Examples

Since Pleiades consists of three different processor types, Harpertown, Nehalem-EP and Westmere-EP, keep the following in mind when requesting PBS resources for your job:

- Starting May 1, 2011, charging on the usage of the three Pleiades processor types is based on a common Standard Billing Unit which is on a per-node basis. The SBU rate for each of the Pleiades processor type is:

```
Processor Type         SBU Rate (per node)
Westmere-EP            1    (12 cores in a node)
Nehalem-EP             0.8  (8 cores in a node)
Harpertown             0.45 (8 cores in a node)
```

The actural amount of memory per node through PBS is slightly less, 7.6 GB/node for Harpertown and 22.5 GB/node for Nehalem-EP and Westmere-EP.

Use the "model=[har,neh,wes]" attribute to request the processor type(s) for your job. If the processor type is not specified in user's PBS resource list, the job is routed to use the default processor type, Harpertown.

Example 1:

Here are some examples of requesting certain processor models for a 128-process job:

```
#PBS -l select=16:ncpus=8:model=har
# to run all 8 cores on each of 16 Harpertown nodes

#PBS -l select=32:ncpus=4:model=har
# to run on only 4 cores on each of 32 Harpertown nodes
# (note: will be charged for 32 nodes = 256 cores)

#PBS -l select=16:ncpus=8:model=neh
# to run all 8 cores on each of 16 Nehalem-EP nodes

#PBS -l select=11:ncpus=12:model=wes
# to run all 12 cores on each of 11 Westmere-EP nodes
# (4 cores in 11th node will go unused)
```

Note that you can specify both the queue type (-q normal, debug, long, wide, low) and the processor type (-l model=har,neh,wes). For example:

```
#PBS -q normal
#PBS -l select=16:ncpus=8:model=neh
```

If your application can run on any of the three processor types, you may want to submit your job to a processor type that has more unoccupied nodes by other running jobs. This can possibly reduce the wait time of your job. The script *node_stats.sh* provides information about the total, used and free nodes for each processor type. For example:

```
%/u/scicon/tools/bin/node_stats.sh


 Pleiades Nodes Total: 9394
 Pleiades Nodes Used : 8128
 Pleiades Nodes Free : 1266

 Harpertown Total: 5854 Used: 4878 Free: 976
 Nehalem    Total: 1255 Used: 1036 Free: 219
 Westmere   Total: 2285 Used: 2214 Free: 71

Currently queued jobs are requesting: 1463 Harpertown, 1767 Nehalem,
2860 Westmere nodes
```

```
Add "/u/scicon/tools/bin" to your PATH in .cshrc or other
shell start up scripts to avoid having to type in the complete
path for this tool.
```

You can also find for each job what processor models are used by looking at the "Model" field of the output of the command:

```
%qstat -a -W o=+model
```

- The Harpertown nodes in rack 32 have 16 GB memory/node instead of 8 GB/node.

Example 2:

This example shows a request of 2 nodes with bigmem in rack 32.

```
#PBS -l select=2:ncpus=8:bigmem=true:model=har
```

- For a multi-node PBS job, the NCPUs used in each node can be different. This is useful for jobs that need more memory for some processes, but less for other processes. Resource requests can be done in "chunks" for a job with varying NCPUs per node.

Example 3:

This example shows a request of two chunks of resources. In the first chunk, 1 CPU in 1 node, and in the second chunk, 8 CPUs in each of three other nodes are requested:

```
#PBS -l select=1:ncpus=1+3:ncpus=8
```

- A PBS job can run across different processor types. This can be useful in two scenarios:

1. when you can not find enough free nodes within one model for your job
2. when some of your processes need more memory while others need much less

This can be accomplished by specifying the resources in "chunks", with one chunk asking for one processor type while another chunk asking for a different processor type.

Example 4

Here is an example to request 1 Westmere node (which provides 24 GB/node) and 3 Harpertown nodes (which provides 8 GB/node).

```
#PBS -lplace=scatter:excl:group=model
#PBS -lselect=1:ncpus=12:mpiprocs=12:model=wes+3:ncpus=8:mpiprocs=8:model=har
```

Default Variables Set by PBS

DRAFT

This article is being reviewed for completeness and technical accuracy.

You can use the "env" command--either in a PBS script or on the command line of a PBS interactive session--to find out what environment variables are set within a PBS job. In addition to the PBS_xxxx variables, the following two are useful to know:

- NCPUS defaults to number of CPUs that you requested for the node.
- OMP_NUM_THREADS defaults to 1 unless you explicitly set it to a different number.

  If your PBS job runs an OpenMP or MPI/OpenMP application, this variable sets the number of threads in the parallel region.
- FORT_BUFFERED defaults to 1.

  Setting this variable to 1 enables records to be accumulated in the buffer and flushed to disk later.

Sample PBS Script for Pleiades

DRAFT

This article is being reviewed for completeness and technical accuracy.

```
#PBS -S /bin/csh
#PBS -N cfd
# This example uses the Harpertown nodes
# User job can access ~7.6 GB of memory per Harpertown node.
# A memory intensive job that needs more than ~0.9 GB
# per process should use less than 8 cores per node
# to allow more memory per MPI process. This example
# asks for 64 nodes and 4 MPI processes per node.
# This request implies 64x4 = 256 MPI processes for the job.
#PBS -l select=64:ncpus=8:mpiprocs=4:model=har
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=a0801
#PBS -m e

# Currently, there is no default compiler and MPI library set.
# You should load in the version you want.
# Currently, MVAPICH or SGI's MPT are available in 64-bit only,
# you should use a 64-bit version of the compiler.

module load comp-intel/11.1.046
module load mpi-sgi/mpt.2.04.10789

# By default, PBS executes your job from your home directory.
# However, you can use the environment variable
# PBS_O_WORKDIR to change to the directory where
# you submitted your job.

cd $PBS_O_WORKDIR

# use of dplace to pin processes to processors may improve performance
# Here you request to pin processes to processors 2, 3, 6, 7 of each node.
# This helps for using the Harpertown nodes, but not for Nehalem-EP or
# Westmere-EP nodes

# The resource request of select=64 and mpiprocs=4 implies
# that you want to have 256 MPI processes in total.
# If this is correct, you can omit the -np 256 for mpiexec
# that you might have used before.

mpiexec dplace -s1 -c2,3,6,7 ./grinder < run_input > output

# It is a good practice to write stderr and stdout to a file (ex: output)
# Otherwise, they will be written to the PBS stderr and stdout in /PBS/spool,
# which has limited amount  of space. When /PBS/spool is filled up, any job
# that tries to write to /PBS/spool will die.
# -end of script-
```

Pleiades devel Queue

NAS provides a special *devel* queue that provides faster turnaround when doing development work.

Currently, 512 Westmere nodes are reserved for the Pleiades *devel* queue, 24x7. The maximum wall-time allowed is 2:00:00 and the maximum NCPUS is 4800. Each user is allowed to have only one job running in the *devel* queue at any one time.

To improve PBS job scheduling response time, the *devel* queue and its resources (for example, nodes) have been moved to a second PBS server (pbspl3). With this move, users must specify the server name for jobs managed by pbspl3 with qsub, qstat, and qdel if the command is launched from a Pleiades front-end node (pfe[1-12] or bridge[1-4]). For example:

```
pfe1% qsub -q devel@pbspl3 job_script
1234.pbspl3.nas.nasa.gov

pfe1% qstat devel@pbspl3

pfe1% qstat -a @pbspl3

pfe1% qstat -u zsmith @pbspl3

pfe1% qstat 1234.pbspl3

pfe1% qdel 1234.pbspl3
```

Alternatively, if you set the environment variable **PBS_DEFAULT** to pbspl3, you can skip pbspl3 in your qsub, qstat, qdel commands. For example (in csh):

```
pfe1% setenv PBS_DEFAULT pbspl3

pfe1% qsub -q devel job_script
1234.pbspl3.nas.nasa.gov

pfe1% qstat devel

pfe1% qstat -a

pfe1% qstat -u zsmith

pfe1% qstat 1234

pfe1% qdel 1234
```

Use the csh command *unsetenv PBS_DEFAULT* to return to using the default PBS server, pbspl1.

Note that the changes described here do not apply to jobs submitted to the other queues (normal, long, debug, and all special queues) served by the default server, pbspl1.

To see all jobs you have submitted to pbspl1 or pbspl3 (using username zsmith in the example below), type the following:

```
pfe1% qstat @pbspl1 @pbspl3 -W combine -u zsmith
```

PBS on Columbia

Overview

DRAFT

This article is being reviewed for completeness and technical accuracy.

On Columbia, PBS (version 9.2) is used to manage batch jobs that run on the four compute systems (Columbia21-24). PBS features that are common to all NAS systems are described in other articles. Read the following articles for Columbia-specific PBS information:

- queue structure

- resource request examples
- default variables set by PBS
- sample PBS scripts

Resources Request Examples

DRAFT

This article is being reviewed for completeness and technical accuracy.

All of the Columbia compute engines, Columbia21-24, are single system image Altix 4700 systems:

```
Columbia21 (508 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia22 (2044 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia23 (1020 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia23 (1020 CPUs total, 1.8 GB memory/CPU through PBS)
```

Here are a few examples of requesting resources on Columbia:

Example 1:

If your job needs fewer than 508 CPUs and you do not care which Columbia system to run your job on, simply use *ncpus* to specify the number of CPUs that you want for your job. For example:

```
#PBS -l ncpus=256
```

Example 2:

If you specify both the *ncpus* and *mem* for your job, PBS will make sure that your job is allocated enough resources to satisfy both *ncpus* and *mem*. For example, if you request 4 CPUs and 14 GB of memory, your job will be allocated 8 CPUs and 14.4 GB because the amount of memory associated with 4 CPUs is not enough to satisfy your memory request.

```
#PBS -l ncpus=4,mem=14GB
```

Example 3:

If you want your job to run on a specific Columbia machine, for example, Columbia22 with 256 CPUs, use

```
#PBS -l select=host=columbia22:ncpus=256
```

Note that the ncpus request must appear with the select=host request and must not be present as a separate request either on the qsub command line or in the PBS script.
Example 4:

If you ever need to run a job across two Columbia systems, for example, 508 CPUs on one Columbia and another 508 CPUs on another, use

```
#PBS -l select=2:ncpus=508,place=scatter
```

Default Variables Set by PBS

DRAFT

This article is being reviewed for completeness and technical accuracy.

You can use the "env" command--either in a PBS script or from the command line of an interactive PBS session--to find out what environment variables are set within a PBS job. In addition to the PBS_xxxx variables, the following ones are useful to know.

- NCPUS defaults to number of CPUs that you requested.
- OMP_NUM_THREADS defaults to 1 unless you explicitly set it to a different number.

  If your PBS job runs an OpenMP or MPI/OpenMP application, this variable sets the number of threads in the parallel region.
- OMP_DYNAMIC defaults to *false*.

  If your PBS job runs an OpenMP application, this disables dynamic adjustment of the number of threads available for execution of parallel regions.
- MPI_DSM_DISTRIBUTE defaults to *true*.

  If your PBS job runs an MPI application, this ensures that each MPI process gets a unique CPU and physical memory on the node with which that CPU ist is associated.
- FORT_BUFFERED defaults to 1.

  Setting this variable to 1 enables records to be accumulated in the buffer and flushed to disk later.

Sample PBS Script for Columbia

DRAFT

This article is being reviewed for completeness and technical accuracy.

```
#PBS -S /bin/csh
#PBS -N cfd
#PBS -l ncpus=4
#PBS -l mem=7776MB
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=g12345
#PBS -m e

# By default, PBS executes your job from your home directory.
# However, you can use the environment variable
# PBS_O_WORKDIR to change to the directory where
# you submitted your job.

cd $PBS_O_WORKDIR

# For MPI jobs, there is an SGI MPT module loaded by default, unless you
# modify your shell start up script to unload it or switch to a different
# version. You can use either mpiexec or mpirun to start your job.

mpiexec -np 4 ./a.out < input > output

# It is a good practice to write stderr and stdout to a file (ex: output)
# Otherwise, they will be written to the PBS stderr and stdout in /PBS/spool
# which has limited amount  of space. When /PBS/spool is filled up, any job
# that tries to write to /PBS/spool will die.

# -end of script-
```

Troubleshooting PBS Jobs

Common Reasons for Being Unable to Submit Jobs

DRAFT

This article is being reviewed for completeness and technical accuracy.

There are several common reasons why you might not be able to successfully submit a job to PBS:

- Resource request exceeds resource limits

  qsub: Job exceeds queue resource limits

  Reduce your resource request to below the limit or use a different queue.
- AUID or GID not authorized to use a specific queue

  If you get the following message after submitting a PBS job:

  qsub: Unauthorized Request

  it is possible that you tried submitting to a queue which is accessible only to certain groups or users. You can check the "qstat -fQ" output and see if there is an acl_groups or a acl_users list. If your group or username is not in the lists, you will have to submit to a different queue."

- AUID not authorized to use a specific GID

  If you get the following message after submitting a PBS job:

  qsub: Bad GID for job execution

  it is possible that your AUID has not been added to use allocations under a specific GID. Please consult with the principal investigator of that GID and ask him/her to submit a request to support@nas.nasa.gov to add your AUID under that GID.
- Queue is disabled

  If you get the following message after submitting a PBS job

  qsub: Queue is not enabled

  you should submit to a different queue which is enabled.
- Not enough allocation left

An automated script is used to check if a GID is over its allocation everyday. If it does, that GID is removed from PBS access control list and users of that GID will not be able to submit jobs.

Users can check the amount of allocations remaining using the acct_ytd command. In addition, if you see in your PBS output file some message regarding your GID allocation usage is near its limit or is already over, ask your PI to request for more allocation.

Common Reasons Why Jobs Won't Start

Once you've successfully submitted your job, there may be several common reasons why it might not run:

- **The job is waiting for resources**

  Your job may be waiting for resources, due to one of the following:
  - All resources are tied up with running jobs, and no other jobs can be started.
  - PBS may have enough resources to run your job, however, there is another higher priority job that needs more resources than what is available, and PBS is draining the system (including not running any new jobs) in order to accommodate the high-priority job.
  - Some users submit too many jobs at once (e.g., more than 100), and the PBS scheduler becomes swamped with sorting jobs and is not able to start jobs effectively.
  - In the case when you request your job to run on a specific node or group of nodes, your job is likely to wait in the queue longer than if you do not request specific nodes.
- **Your mission share has run out**

  Your mission shares have been used up. The available resources that you saw belong to other missions, which can be borrowed. However, your job may have requested a wall-time that is too long (more than 4 hours for Pleiades), which prevents your job from borrowing the resources.

  See also, Mission Shares Policy on Pleiades.
- **The system is going into dedicated time**

  When dedicated time (DED) is scheduled for hardware and/or software work, the PBS scheduler will not start a job if the projected end time runs past the beginning of the DED time. If you are able to reduce the requested wall-time so that your job will finish running prior to DED time, then your job can then be considered for running. To change the wall-time request for your job, follow the example below :

  ```
  %qalter -l walltime=hh:mm:ss jobid
  ```
- **Scheduling is turned off**

  Sometimes job scheduling is turned off by control room staff or a system administrator. This is usually done when there are system or PBS issues that need to be resolved before jobs can be scheduled to run. When this happens, you should see the following message near the beginning of the "qstat -au your_userid" output.

  ```
  +++Scheduling turned off.
  ```
- **Your job has been placed in "H" mode**

A job can be placed on hold either by the job owner or by someone who has root privilege, such as a system administrator. If your job has been placed on hold by a system administrator, you should get an email explaining the reason for the hold.

- **Your home filesystem or default /nobackup filesystem is down**

  When a PBS job starts, the PBS prologue checks to determine whether your home filesystem and default /nobackup are available before executing the commands in your script. If your default /nobackup filesystem is down, PBS can not run your job and it will put the job back in the queue. If your PBS job does not need any file in that filesystem, you can tell PBS that your job will not use the default /nobackup so that your job can start running. For example, if your default is /nobackupp10 (for Pleiades), you can add the following in your PBS script:

  ```
  #PBS -l /nobackupp10=0
  ```

Using pdsh_gdb for Debugging Pleiades PBS Jobs

DRAFT

This article is being reviewed for completeness and technical accuracy.

A script called *pdsh_gdb*, created by NAS staff Steve Heistand, is available on Pleiades under */u/scicon/tools/bin* for debugging PBS jobs **while the job is running**.

Launching this script from a Pleiades front-end node allows one to connect to each compute node of a PBS job and create a stack trace of each process. The aggregated stack trace from each process will be written to a user specified directory (by default, it is written to ~/tmp).

Here is an example of how to use this script:

```
pfe1% mkdir tmp
pfe1% /u/scicon/tools/bin/pdsh_gdb -j jobid -d tmp -s -u nas_username
```

More usage information can be found by launching pdsh_gdb without any option:

```
pfe1% /u/scicon/tools/bin/pdsh_gdb
```

Effective Use of PBS

Streamlining File Transfers from Pleiades Compute Nodes to Lou

Some users prefer to streamline the storage of files (created during a job run) to Lou, within a PBS job. Since Pleiades compute nodes do not have network access to the outside world, all file transfers to Lou within a PBS job must go through the front-ends (pfe[1-12], bridge[1,2]) first.

Here is an example of what you can add to your PBS script to accomplish this:

1. Ssh to a front-end node (for example, bridge2) and create a directory on Lou where the files are to be copied.

   ```
   ssh -q bridge2 "ssh -q lou mkdir -p $SAVDIR"
   ```
   Here, $SAVDIR is assumed to have been defined earlier in the PBS script. Note the use of *-q* for quiet-mode, and double quotes so that shell variables are expanded prior to the *ssh* command being issued.

2. Use scp via bridge[1,2] to transfer the files.

   ```
   ssh -q bridge2 "scp -q $RUNDIR/* lou:$SAVDIR"
   ```
   Here, $RUNDIR is assumed to have been defined earlier in the PBS script.

Avoiding Job Failure from Overfilling /PBS/spool

Before a PBS job is completed, its error and output files are kept in the /PBS/spool directory of the first node of your PBS job. The space under /PBS/spool is limited, however, and when it fills up, any job that tries to write to /PBS/spool may die. To prevent this, you should *not* write large amount of contents in the PBS output/error files.

If your executable normally produces a lot of output to the screen, you should redirect its output in your PBS script. For example:

```
#PBS ...
mpiexec a.out > output
```

To see the contents of your PBS output/error files before your job completes, follow the two steps below:

1. Find out the first node of your PBS job using "-W o=+rank0" for qstat:

   ```
   %qstat -u your_username -W o=+rank0
   JobID         User   Queue  Jobname   TSK Nds    wallt S    wallt  Eff Rank0
   ------------- ------ ------ -------- ---- --- -------- - -------- ---- ---------
   868819.pbspl1 zsmith long   ABC       512  64 5d+00:00 R 3d+08:39 100% r162i0n14
   ```

   This shows that the first node is r162i0n14.

2. Log in to the first node and *cd* to /PBS/spool to find your PBS stderr/out file(s). You can view the content of these files using *vi* or *view*.

```
%ssh r162i0n14
%cd /PBS/spool
%ls -lrt
-rw------- 1 zsmith a0800 49224236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.OU
-rw------- 1 zsmith a0800  1234236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.ER
```

Running Multiple Serial Jobs to Reduce Walltime

DRAFT

This article is being reviewed for completeness and technical accuracy.

On Pleiades, running multiple serial jobs within a single batch job can be accomplished with following example PBS scripts. The maximum number of processes you can run on a single node will be limited to the core-count-per-node or the maximum number that will fit in a given node's memory, whichever is smaller.

```
processor type  cores/node      available memory/node
 Harpertown          8                    7.6 GB
 Nehalem-EP          8                   22.5 GB
 Westmere-EP        12                   22.5 GB
```

The examples below allow you to spawn serial jobs accross nodes using the mpiexec command. Note that a special version of mpiexec from the mpi-mvapich2/1.4.1/intel module is needed in order for this to work. This mpiexec keeps track of $PBS_NODEFILE and places each serial job onto the CPUs listed in $PBS_NODEFILE properly. The use of the arguments "-comm none" for this version of mpiexec is essential for serial codes or scripts. In addition, to launch multiple copies of the serial job at once, the use of the mpiexec-supplied $MPIEXEC_RANK environment variable is needed to distinguish different input/output files for each serial job. This is demonstrated with the use of a wrapper script "wrapper.csh" in which the input/output identifier (i.e., ${rank}) is calculated from the sum of $MPIEXEC_RANK and an argument provided as input by the user.

Example 1:

This first example runs 64 copies of a serial job, assuming that 4 copies will fit in the available memory on one node and 16 nodes are used.

serial1.pbs:

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=16:ncpus=4
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

mpiexec -comm none -np 64 wrapper.csh 0
```

wrapper.csh:

```
#!/bin/csh -f
```

```
@ rank = $1 + $MPIEXEC_RANK
./a.out < input_${rank}.dat > output_${rank}.out
```

This example assumes that input files are named input_0.dat, input_1.dat, ... and that they are all located in the directory where the PBS script is submitted from (i.e., $PBS_O_WORKDIR). If the input files are in different directories, then wrapper.csh can be modified appropriately to cd into different directories as long as the directory names are differentiated by a single number that can be obtained from $MPIEXEC_RANK (=0, 1, 2, 3, ...). In addition, be sure that wrapper.csh is executable by you and you have the current directory included in your path.

Example 2:

A second example provides the flexibility where the total number of serial jobs may not be the same as the total number of CPUs requested in a PBS job. Thus, the serial jobs are divided into a few batches and the batches are processed sequentially. Again, the wrapper script is used where multiple versions of the program "a.out" in a batch are run in parallel.

serial2.pbs:

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=10:ncpus=3
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

# This will start up 30 serial jobs 3 per node at a time.
# There are 64 jobs to be run total, only 30 at a time.

# The number to run in total defaults here to 64 or the value
# of PROCESS_COUNT that is passed in via the qsub line like:
# qsub -v PROCESS_COUNT=48 serial2.pbs
#

# the total number to run at once is automatically determined
# at runtime by the number of cpus available.
# qsub -v PROCESS_COUNT=48 -l select=4:ncpus=3 serial2.pbs
# would make this 12 per pass not 30. no changes to script needed.

if ( $?PROCESS_COUNT ) then
 set total_runs=$PROCESS_COUNT
else
 set total_runs=64
endif

set batch_count=`wc -l < $PBS_NODEFILE`

set count=0
```

```
while ($count < $total_runs)
  @ rank_base = $count
  @ count += $batch_count
  @ remain = $total_runs - $count
  if ($remain < 0) then
    @ run_count = $total_runs % $batch_count
  else
    @ run_count = $batch_count
  endif
  mpiexec -comm none -np $run_count wrapper.csh $rank_base
end
```

Checking the Time Remaining in a PBS Job from a Fortran Code

DRAFT

This article is being reviewed for completeness and technical accuracy.

During job execution, sometimes it is useful to find out the amount of time remaining for your PBS job. This allows you to decide if you want to gracefully dump restart files and exit before PBS kills the job.

If you have an MPI code, you can call MPI_WTIME and see if the elapsed walltime has exceeded some threshold to decide if the code should go into the shutdown phase.

For example,

```
include "mpif.h"

real (kind=8) :: begin_time, end_time

begin_time=MPI_WTIME()
do work
end_time = MPI_WTIME()

if (end_time - begin_time > XXXXX) then
   go to shutdown
endif
```

In addition, the following library has been made available on Pleiades for the same purpose:

/u/scicon/tools/lib/pbs_time_left.a

To use this library in your Fortran code, you need to:

1. Modify your Fortran code to define an external subroutine and an integer*8 variable

   ```
   external pbs_time_left
   integer*8 seconds_left
   ```
2. Call the subroutine in the relevant code segment where you want the check to be performed

   ```
   call pbs_time_left(seconds_left)
   print*,"Seconds remaining in PBS job:",seconds_left


   The return value from pbs_time_left is only  accurate to within a minute or
   ```

3. Compile your modified code and link with the above library using, for example

   ```
   LDFLAGS=/u/scicon/tools/lib/pbs_time_left.a
   ```

Best Practices

Streamlining File Transfers from Pleiades Compute Nodes to Lou

Some users prefer to streamline the storage of files (created during a job run) to Lou, within a PBS job. Since Pleiades compute nodes do not have network access to the outside world, all file transfers to Lou within a PBS job must go through the front-ends (pfe[1-12], bridge[1,2]) first.

Here is an example of what you can add to your PBS script to accomplish this:

1. Ssh to a front-end node (for example, bridge2) and create a directory on Lou where the files are to be copied.

   ```
   ssh -q bridge2 "ssh -q lou mkdir -p $SAVDIR"
   ```
   Here, $SAVDIR is assumed to have been defined earlier in the PBS script. Note the use of *-q* for quiet-mode, and double quotes so that shell variables are expanded prior to the *ssh* command being issued.

2. Use scp via bridge[1,2] to transfer the files.

   ```
   ssh -q bridge2 "scp -q $RUNDIR/* lou:$SAVDIR"
   ```
   Here, $RUNDIR is assumed to have been defined earlier in the PBS script.

Increasing File Transfer Rates

One challenge users face is moving large amounts of data efficiently to/from NAS across the network. Often, minor system, software, or network configuration changes can increase network performance an order of magnitude or more. This article describes some methods for increasing data transfer performance.

If you are experiencing slow transfer rates, try these quick tips:

- Transfer using the bridge nodes (bridge1, bridge2) instead of the Pleiades front-end systems (PFEs). The bridge nodes have much more memory, along with 10-Gigabit Ethernet interfaces to accommodate many large transfers. The PFEs often become oversubscribed and cause slowness.
- If using the scp command, make sure you are using OpenSSH version 5 or later. Older versions of SSH have a hard limit on transfer rates and are not designed for WAN transfers. You can check your version of SSH by running the command ssh -V.
- For large files that are a gigabyte or larger, we recommend using BBFTP. This application allows for transferring simultaneous streams of data and doesn't have the overhead of encrypting all the data (authentication is still encrypted).

Online Network Testing Tools

The NAS PerfSONAR Service provides a custom website that that allows you to quickly self-diagnose your remote network connection issues, and reports the maximum bandwidth between sites, as well as any problems in the network path. Command-line tools are available if your system does not have a web browser.

Test results are also sent to our network experts, who will analyze traffic flows, identify problems, and work to resolve any bottlenecks that limit your network performance, whether the problem is at NAS or a remote site.

One-on-One Help

If you still require assistance in increasing your file transfer rates, please contact the NAS Control Room at support@nas.nasa.gov, and a network expert will work with you or your local administrator one-on-one to identify methods for increasing your rates.

To learn about other network-related support areas. see also, End-to-End Networking Services.

Effective Use of Resources with PBS

Streamlining File Transfers from Pleiades Compute Nodes to Lou

Some users prefer to streamline the storage of files (created during a job run) to Lou, within a PBS job. Since Pleiades compute nodes do not have network access to the outside world, all file transfers to Lou within a PBS job must go through the front-ends (pfe[1-12], bridge[1,2]) first.

Here is an example of what you can add to your PBS script to accomplish this:

1. Ssh to a front-end node (for example, bridge2) and create a directory on Lou where the files are to be copied.

   ```
   ssh -q bridge2 "ssh -q lou mkdir -p $SAVDIR"
   ```
   Here, $SAVDIR is assumed to have been defined earlier in the PBS script. Note the use of *-q* for quiet-mode, and double quotes so that shell variables are expanded prior to the *ssh* command being issued.

2. Use scp via bridge[1,2] to transfer the files.

   ```
   ssh -q bridge2 "scp -q $RUNDIR/* lou:$SAVDIR"
   ```
   Here, $RUNDIR is assumed to have been defined earlier in the PBS script.

Avoiding Job Failure from Overfilling /PBS/spool

Before a PBS job is completed, its error and output files are kept in the /PBS/spool directory of the first node of your PBS job. The space under /PBS/spool is limited, however, and when it fills up, any job that tries to write to /PBS/spool may die. To prevent this, you should *not* write large amount of contents in the PBS output/error files.

If your executable normally produces a lot of output to the screen, you should redirect its output in your PBS script. For example:

```
#PBS ...
mpiexec a.out > output
```

To see the contents of your PBS output/error files before your job completes, follow the two steps below:

1. Find out the first node of your PBS job using "-W o=+rank0" for qstat:

   ```
   %qstat -u your_username -W o=+rank0
   JobID         User   Queue  Jobname   TSK Nds    wallt S    wallt  Eff Rank0
   ------------- ------ ------ -------- ---- --- -------- - -------- ---- ---------
   868819.pbspl1 zsmith long   ABC       512  64 5d+00:00 R 3d+08:39 100% r162i0n14
   ```

   This shows that the first node is r162i0n14.

2. Log in to the first node and *cd* to /PBS/spool to find your PBS stderr/out file(s). You can view the content of these files using *vi* or *view*.

```
%ssh r162i0n14
%cd /PBS/spool
%ls -lrt
-rw------- 1 zsmith a0800 49224236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.OU
-rw------- 1 zsmith a0800  1234236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.ER
```

Running Multiple Serial Jobs to Reduce Walltime

DRAFT

This article is being reviewed for completeness and technical accuracy.

On Pleiades, running multiple serial jobs within a single batch job can be accomplished with following example PBS scripts. The maximum number of processes you can run on a single node will be limited to the core-count-per-node or the maximum number that will fit in a given node's memory, whichever is smaller.

```
processor type  cores/node      available memory/node
 Harpertown         8                     7.6 GB
 Nehalem-EP         8                    22.5 GB
 Westmere-EP       12                    22.5 GB
```

The examples below allow you to spawn serial jobs accross nodes using the mpiexec command. Note that a special version of mpiexec from the mpi-mvapich2/1.4.1/intel module is needed in order for this to work. This mpiexec keeps track of $PBS_NODEFILE and places each serial job onto the CPUs listed in $PBS_NODEFILE properly. The use of the arguments "-comm none" for this version of mpiexec is essential for serial codes or scripts. In addition, to launch multiple copies of the serial job at once, the use of the mpiexec-supplied $MPIEXEC_RANK environment variable is needed to distinguish different input/output files for each serial job. This is demonstrated with the use of a wrapper script "wrapper.csh" in which the input/output identifier (i.e., ${rank}) is calculated from the sum of $MPIEXEC_RANK and an argument provided as input by the user.

Example 1:

This first example runs 64 copies of a serial job, assuming that 4 copies will fit in the available memory on one node and 16 nodes are used.

serial1.pbs:

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=16:ncpus=4
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

mpiexec -comm none -np 64 wrapper.csh 0
```

wrapper.csh:

```
#!/bin/csh -f
```

```
@ rank = $1 + $MPIEXEC_RANK
./a.out < input_${rank}.dat > output_${rank}.out
```

This example assumes that input files are named input_0.dat, input_1.dat, ... and that they are all located in the directory where the PBS script is submitted from (i.e., $PBS_O_WORKDIR). If the input files are in different directories, then wrapper.csh can be modified appropriately to cd into different directories as long as the directory names are differentiated by a single number that can be obtained from $MPIEXEC_RANK (=0, 1, 2, 3, ...). In addition, be sure that wrapper.csh is executable by you and you have the current directory included in your path.

Example 2:

A second example provides the flexibility where the total number of serial jobs may not be the same as the total number of CPUs requested in a PBS job. Thus, the serial jobs are divided into a few batches and the batches are processed sequentially. Again, the wrapper script is used where multiple versions of the program "a.out" in a batch are run in parallel.

serial2.pbs:

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=10:ncpus=3
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

# This will start up 30 serial jobs 3 per node at a time.
# There are 64 jobs to be run total, only 30 at a time.

# The number to run in total defaults here to 64 or the value
# of PROCESS_COUNT that is passed in via the qsub line like:
# qsub -v PROCESS_COUNT=48 serial2.pbs
#

# the total number to run at once is automatically determined
# at runtime by the number of cpus available.
# qsub -v PROCESS_COUNT=48 -l select=4:ncpus=3 serial2.pbs
# would make this 12 per pass not 30. no changes to script needed.

if ( $?PROCESS_COUNT ) then
 set total_runs=$PROCESS_COUNT
else
 set total_runs=64
endif

set batch_count=`wc -l < $PBS_NODEFILE`

set count=0
```

```
while ($count < $total_runs)
  @ rank_base = $count
  @ count += $batch_count
  @ remain = $total_runs - $count
  if ($remain < 0) then
    @ run_count = $total_runs % $batch_count
  else
    @ run_count = $batch_count
  endif
  mpiexec -comm none -np $run_count wrapper.csh $rank_base
end
```

Checking the Time Remaining in a PBS Job from a Fortran Code

DRAFT

This article is being reviewed for completeness and technical accuracy.

During job execution, sometimes it is useful to find out the amount of time remaining for your PBS job. This allows you to decide if you want to gracefully dump restart files and exit before PBS kills the job.

If you have an MPI code, you can call MPI_WTIME and see if the elapsed walltime has exceeded some threshold to decide if the code should go into the shutdown phase.

For example,

```
include "mpif.h"

real (kind=8) :: begin_time, end_time

begin_time=MPI_WTIME()
do work
end_time = MPI_WTIME()

if (end_time - begin_time > XXXXX) then
   go to shutdown
endif
```

In addition, the following library has been made available on Pleiades for the same purpose:

/u/scicon/tools/lib/pbs_time_left.a

To use this library in your Fortran code, you need to:

1. Modify your Fortran code to define an external subroutine and an integer*8 variable

   ```
   external pbs_time_left
   integer*8 seconds_left
   ```
2. Call the subroutine in the relevant code segment where you want the check to be performed

   ```
   call pbs_time_left(seconds_left)
   print*,"Seconds remaining in PBS job:",seconds_left


   The return value from pbs_time_left is only  accurate to within a minute or
   ```

3. Compile your modified code and link with the above library using, for example

   ```
   LDFLAGS=/u/scicon/tools/lib/pbs_time_left.a
   ```

Memory Usage on Pleiades

Memory Usage Overview

Running jobs on cluster systems such as Pleiades requires more attention to the memory usage of a job than on shared memory systems. Below are a few factors that limit the amount of memory available to your running job:

- The total physical memory of a Pleiades compute node varies from 8 GB to 24 GB. A small amount of the physical memory is used by the system kernel. Through PBS, a job can access up to about 7.6 GB of an 8-GB node (Harpertown) and about 22.5 GB of a 24-GB node (Nehalem-EP and Westmere-EP).
- The PBS prologue tries to clean up the memory used by the previous job that ran on the nodes of your current running job. If there is a delay in flushing the previous job's data from memory to disks (for example, due to Lustre issues), the actual amount of free memory available to your job will be less.
- I/O uses buffer cache that also occupies memory. If your job does a large amount of I/O, the amount of memory left for your running processes will be less.

If your job uses more than 1 node, beware that the memory usage reported in the PBS output file is not the total memory usage for your job: rather, it is the *memory used in the first node* of your job. To help you get a more accurate picture of the memory usage of your job, we provide a few in-house tools, listed below.

1. ***qtop.pl*** invokes *top* on the compute nodes of a job, and provides a snapshot of the amount of used and free memory of the whole node and the amount used by each running process. For more information, read the article Checking Memory Usage of a Batch Job Using qtop.pl.
2. ***qps*** invokes *ps* on the compute nodes of a job, and provides a snapshot of the %mem used by its running processes. For more information, read the article Checking Memory Usage of a Batch Job Using qps.
3. ***qsh.pl*** can be used to invoke the command *cat /proc/meminfo* on the compute nodes to provide a snapshot of the total and free memory in each node. For more information, read the article Checking Memory Usage of a Batch Job Using qsh.pl and "cat /proc/meminfo".
4. ***gm.x*** and ***gm_post.x*** provide the memory high water mark for each process of your job when the job finishes. For more information, read the article Checking Memory Usage of a Batch Job Using gm.x.

These tools are installed under the directory /u/scicon/tools/bin. It is a good idea to include this directory in your path by modifying your shell startup script so that you don't have to provide the complete path name when using these tools. For example:

```
set path = ( $path /u/scicon/tools/bin )
```

If your job runs out of memory and is killed by the kernel, this event was probably recorded in system log files. Instructions on how to check whether this is the case are provided in the article Checking if a Job was Killed by the OOM Killer.

If your job needs more memory, read the article How to Get More Memory for your Job for possible approaches.

Checking memory usage of a batch job using qps

User Jeff West provided us with a Perl script called *qps* (available under /u/scicon/tools/bin) that securely connects (via ssh) into each node of a running job and gets process status (*ps)* information on each node.

Syntax:

```
pfe1% qps jobid
```

Example:

```
pfe1% qps 26130

*** Job 26130, User abc, Procs 1
NODE      TIME     %MEM %CPU STAT TASK
r1i0n14 10:17:13  2.8 99.9 RL   ./a.out
r1i0n14 10:17:12  2.9 99.9 RL   ./a.out
r1i0n14 10:17:18  2.9 99.9 RL   ./a.out
r1i0n14 10:16:34  2.9 99.8 RL   ./a.out
r1i0n14 10:17:11  2.9 99.9 RL   ./a.out
r1i0n14 10:17:13  2.9 99.9 RL   ./a.out
r1i0n14 10:17:12  2.9 99.9 RL   ./a.out
r1i0n14 10:17:15  2.9 99.9 RL   ./a.out
```

Note: The % memory usage by a process reported by this script is the percentage of memory in *the whole node*. This script currently works only when users specify `ncpus=8` in the PBS resource request.

If you want to use *qps* to monitor the memory used by a job that requested a number of CPUs other than 8, then make a copy of the qps script and change that single occurrence of '8' on line 95 to the appropriate number of CPUs requested on each node.

Checking memory usage pf a batch job using qtop.pl

DRAFT

This article is being reviewed for completeness and technical accuracy.

A Perl script called *qtop.pl* (available under /u/scicon/tools/bin) was provided by Bob Hood of the NAS staff. This script ssh's into the nodes of a PBS job and performs the command *top*. The output of qtop.pl provides memory usage for the whole node and for each process.

Syntax:

```
pfe1% qtop.pl [-b] [-p n] [-P s] [-h n] [-H s] [-t s] [-N s] PBSjobid
      -b   : (for running in background or batch) don't run 'resize' command
      -p n : show at most n processes per host
      -P s : show only procs in s, a comma-separated list of ranges
               e.g. -P 1,8-9
      -h   : don't show the column header line
      -H s : show only header lines in s, comma-separated ranges
               e.g. -H 1-2,7
               e.g. -H 0 (don't show any lines)
      -t s : pass string s (must be one argument) to top command
      -n s : show output only from nodes in s, comma-separated ranges
               e.g. -n 0,2-3          (relative node #'s)
      -N s : show output only from nodes in s, a comma-separated list
               e.g. -N r1i1n14,r1i1n15 (absolute node #'s)
```

Example: to skip the header and list 8 procs per host

```
pfe1% qtop.pl -H 0 -p 8 996093
all nodes in job 996093:  r184i2n12
r184i2n12   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
          20027 zsmith    25   0 23.8g 148m 5320 R  101  0.6   5172:37 a.out
          20028 zsmith    25   0 23.8g 140m 5140 R  101  0.6   5173:35 a.out
          20029 zsmith    25   0 23.9g 286m 6640 R  101  1.2   5172:23 a.out
          20030 zsmith    25   0 23.9g 245m 5040 R  101  1.0   5171:18 a.out
          20031 zsmith    25   0 23.9g 265m 6040 R  101  1.1   5171:46 a.out
          20032 zsmith    25   0 23.9g 246m 5300 R  101  1.0   5171:00 a.out
          20033 zsmith    25   0 23.8g 158m 5476 R  101  0.7   5172:41 a.out
          20034 zsmith    25   0 23.8g 148m 5280 R  101  0.6   5173:02 a.out
```

Checking memory usage of a batch job using qsh.pl and "cat /proc/meminfo"

DRAFT

This article is being reviewed for completeness and technical accuracy.

A Perl script called *qsh.pl* (available under /u/scicon/tools/bin) was provided by NAS staff member Bob Hood. This script ssh's into all the nodes used by a PBS job and runs a command that you supply.

Syntax:

```
pfe1% qsh.pl pbs_jobid command
```

One good use of this script is to check the amount of free memory in the nodes of your PBS job.

Example:

```
pfe1% qsh.pl 30329 "cat /proc/meminfo"

running "cat /proc/meminfo" on:  r56i2n14 r56i2n15
r56i2n14 :
  MemTotal:      8079728 kB
  MemFree:        857936 kB
  Buffers:             0 kB
  Cached:        3775472 kB
...
r56i2n15 :
  MemTotal:      8079728 kB
  MemFree:       5840920 kB
  Buffers:             0 kB
  Cached:         784280 kB
...
```

Checking memory usage of a batch job using gm.x

DRAFT

This article is being reviewed for completeness and technical accuracy.

NAS staff member Henry Jin created a tool called *gm.x* (available under /u/scicon/tools/bin) that reports the memory usage at the end of a run from each process.

Add */u/scicon/tools/bin* to your $PATH so that you can invoke *gm.x* without the full path.

Use the -h option to find out what types of memory usage can be reported:

```
pfe1%gm.x -h
gm - version 1.0
usage: gm.x [-opts] a.out [args]
   -hwm     ; high water mark (VmHWM)
   -rss     ; resident memory size (VmRSS)
   -wrss    ; weighted memory size (WRSS)
   -v       ; verbose flag
Default is by environment variable GM_TYPE (def=WRSS)
```

Note that the -rss option reports the last snapshot of resident set size usage captured by the kernel. With the -wrss option, *gm.x* calls the system function *get_weighted_memory_size*. More information about this function can be found from the man page **man get_weighted_memory_size**.

gm.x can be used for either OpenMP or MPI applications (linked with either SGI's MPT, MVAPICH or Intel MPI libraries) and you do not have to recompile your application for it. A script called *gm_post.x* then takes the per process memory usage information and computes the total memory used and the average memory used per process.

To use *gm.x* for an MPI code, add *gm.x* after the mpiexec options. For example:

```
mpiexec -np 4 gm.x ./a.out
Memory usage for (r1i1n0,pid=9767): 1.458 MB (rank=0)
Memory usage for (r1i1n0,pid=9768): 1.413 MB (rank=1)
Memory usage for (r1i1n0,pid=9770): 1.413 MB (rank=3)
Memory usage for (r1i1n0,pid=9769): 1.417 MB (rank=2)


mpiexec -np 4 gm.x ./a.out | gm_post.x
Number of nodes     = 1
Number of processes = 4
Processes per node  = 4
Total memory        = 5.701 MB

Memory per node     = 5.701 MB
Minimum node memory = 5.701 MB
```

```
Maximum node memory = 5.701 MB

Memory per process  = 1.425 MB
Minimum proc memory = 1.413 MB
Maximum proc memory = 1.458 MB
```

If you use dplace to pin process, add *gm.x* after dplace:

```
mpiexec -np NN dplace -s1 gm.x ./a.out
```

Checking if a Job was Killed by the OOM Killer

If a PBS job runs out of memory and is killed by the Out-Of-Memory (OOM) killer of the kernel, this event is likely (though not always) recorded in system log files. You can confirm this event by checking some of the messages recorded in system log files, and then increase your memory request in order to get your job running.

Follow the steps below to check whether your job has been killed by the OOM killer:

1. Find out when your job ran, what rack numbers were used by your job, and if the job exited with the Exit_status=137 from the tracejob output of your job. For example:

   ```
   pfe[1-12]% ssh pbspl1
   pbspl1% tracejob -n 3 140001
   ```

   where "3" indicates that you want to trace your job (PBS JOBID=140001), which ran within the past 3 days.

2. From the rack numbers (such as r2, r3, ...), you then grep messages that were recorded in the messages file stored in the leader node of those racks for your executable. For example, to look at messages for rack r2:

   ```
   pfe[1-12]% grep abc.exe /net/r2lead/var/log/messages
   Apr 21 00:32:50 r2i2n7 kernel: abc.exe invoked oom-killer:
   gfp_mask=0x201d2, order=0, oomkilladj=-17
   ```

3. Often, the Out-Of-Memory message doesn't make it into the messages file, but will be recorded in a consoles file named by each individual node. For example, to look for abc.exe invoking the OOM killer on node r2i2n7:

   ```
   pfe%  grep abc.exe /net/r2lead/var/log/consoles/r2i2n7
   abc.exe invoked oom-killer: gfp_mask=0x201d2, order=0, oomkilladj=0
   ```

   Note that these messages do not have a timestamp associated with them, so you will need to use an editor to view the file and look for the hourly time markers bracketing when the job ran out of memory. An hourly time marker looks like this:

   ```
   [-- MARK -- Thu Apr 21 00:00:00 2011]
   ```

It's also possible that a system process (such as, pbs_mom or ntpd) is listed as invoking the OOM killer, but it is nevertheless direct evidence that the node had run out of memory.

If you want to monitor the memory use of your job while it is running, you can use the tools listed in the article Memory Usage Overview.

In addition, NAS provides a script called *pbs_oom_check*. This script does the steps above and parses the /var/log/messages on all the nodes associated with pbs_jobid, looking for an instance of OOM killer. The script is available under /u/scicon/tools/bin and works best when run on the host pbspl1.

How to get more memory for your job

DRAFT

This article is being reviewed for completeness and technical accuracy.

If your job was terminated because it needed more memory than what's available in the nodes that it ran on, consider the following:

- Among the Harpertown nodes, the 64 nodes in rack 32 have 16 GB per node instead of 8 GB per node. You can request running your job on rack 32 with the keyword **bigmem=true**. For example, change

  ```
  #PBS -lselect=1:ncpus=8
  ```

  to

  ```
  #PBS -lselect=1:ncpus=8:bigmem=true
  ```
- Run your job on Nehalem-EP or Westmere nodes instead of Harpertown nodes. For example, change

  ```
  #PBS -lselect=1:ncpus=8:model=har
  ```

  to

  ```
  #PBS -lselect=1:ncpus=8:model=neh
  ```

  or

  ```
  #PBS -lselect=1:ncpus=8:model=wes
  ```
- If all processes use about the same amount of memory and you can not fit 8 processes per node (for Harpertown or Nehalem-EP, or 12 processes per node for Westmere-EP), reduce the number of processes per node and request more nodes for your job. For example, change

  ```
  #PBS -lselect=3:ncpus=8:mpiprocs=8:model=neh
  ```

  to

  ```
  #PBS -lselect=6:ncpus=4:mpiprocs=4:model=neh
  ```
- For a typical MPI job where rank 0 does the I/O and uses a lot of buffer cache, assign rank 0 to 1 node by itself. For example, change

```
#PBS -lselect=1:ncpus=8:mpiprocs=8:model=neh
```

to

```
#PBS
-lselect=1:ncpus=1:mpiprocs=1:model=neh+1:ncpus=7:mpiprocs=7:model=neh
```

```
Due to formatting issue, the above may appear as 2 lines. It should
really be just 1 line.
```

- If you suspect that certain nodes that your job ran on had less total physical memory than normal, report it to NAS Help Desk. Those nodes can be offlined and taken care of by NAS staff. This prevents you and other users from using those nodes before they are fixed.

- For certain pre- or post-processing work that needs more than 22.5 GB of memory, run it on the bridge nodes (bridge[1,2]) interactively. Note that jobs running on the bridge nodes can not use more than 48 GB of memory. Also MPI applications that use SGI's MPT library can not run on the bridge nodes.

- For a multi-process or multi-thread job, if any of your processes/threads needs more than 22.5 GB, it won't run on Pleiades. Run it on a shared memory system such as Columbia.

Lustre on Pleiades

Lustre Basics

DRAFT

This article is being reviewed for completeness and technical accuracy.

A Lustre filesystem is a high-performance, shared filesystem (managed with the Lustre software) for Linux clusters. It is highly scalable and can support many thousands of client nodes, petabytes of storage and hundreds of gigabytes per second of I/O throughput.

Main Lustre components:

- Metadata Server (MDS)

  1 or 2 per filesystem; service nodes that manage all metadata operations such as assigning and tracking the names and storage locations of directories and files on the OSTs.
- Metadata Target (MDT)

  1 per filesystem; a storage device where the metadata (name, ownership, permissions and file type) are stored.
- Object Storage Server (OSS)

  1 or multiple per filesystem; service nodes that run the Lustre software stack, provide the actual I/O service and network request handling for the OSTs, and coordinate file locking with the MDS. Each OSS can serve up to ~15 OSTs. The aggregate bandwidth of a Lustre filesystem can approach the sum of bandwidths provided by the OSSes.
- Object Storage Target (OST)

  multiple per filesystem; storage devices where the data in user files are stored. Under Linux 2.6 (current OS on Pleiades), each OST can be up to 8TB in size. Under SLES 11, each OST can be up to 16 GB in size. The capacity of a Lustre filesystem is the sum of the sizes of all OSTs.
- Lustre Clients

  commonly in the thousands per filesystem; compute nodes that mount the Lustre filesystem, and access/use data in the filesystem.

Striping

A user file can be divided into multiple chunks and stored across a subset of the OSTs. The chunks are distributed among the OSTs in a round-robin fashion to ensure load balancing.

Benefits of striping:

- allows one to have a file size larger than the size of an OST
- allows one or more clients to read/write different parts of the same file at the same time and provide higher I/O bandwidth to the file since the bandwidth is aggregated over the multiple OSTs

Drawbacks of striping:

- higher risk of file damage due to hardware malfunction
- increased overhead due to network operations and server contention

There are default stripe configurations for each Lustre filesystem. However, users can set the following stripe parameters for their own directories or files to get optimum I/O performance:

1. stripe_size

   the size of the chunk in bytes; specify with k, m, or g to use units of KB, MB, or GB, respectively; the size must be an even multiple of 65,536 bytes; default is 4MB for all Pleiades Lustre filesystems; one can specify 0 to use the default size.

2. stripe_count

   the number of OSTs to stripe across; default is 1 for most of Pleiades Lustre filesystems (/nobackupp[10-60]); one can specify 0 to use the default count; one can specify -1 to use all OSTs in the filesystem.

3. stripe_offset

   The index of the OST where the first stripe is to be placed; default is -1 which results in random selection; using a non-default value is NOT recommended.

Use the command for setting the stripe parameters:

```
pfe1% lfs setstripe -s stripe_size -c stripe_count -o stripe_offset
dir|filename
```

For example, to create a directory called dir1 with a stripe_size of 4MB and a stripe_count of 8, do

```
pfe1% mkdir dir1
```

```
pfe1% lfs setstripe -s 4m -c 8 dir1
```

Also keep in mind that:

- When a file or directory is created, it will inherit the parent directory's stripe settings.
- The stripe settings of an *existing file* can not be changed. If you want to change the settings of a file, you can create a new file with the desired settings and copy the existing file to the newly created file.

Useful Commands for Lustre

- To list all the OSTs for the filesystem

  ```
  pfe1% lfs osts
  ```

- To list space usage per OST and MDT in human readable format for all Lustre filesystems or for a specific one, for example, /nobackupp10:

  ```
  pfe1% lfs df -h
  pfe1% lfs df -h /nobackupp10
  ```

- To list inode usage for all filesystems or a specific one, for example, /nobackupp10:

  ```
  pfe1% df -i
  pfe1% df -i /nobackupp10
  ```

- To create a new (empty) file or set directory default with specified stripe parameters

  ```
  pfe1% lfs setstripe -s stripe_size -c stripe_count -o
  stripe_offset dir|filename
  ```

- To list the striping information for a given file or directory

  ```
  pfe1% lfs getstripe dir|filename
  ```

- To display disk usage and limits on your /nobackup directory (for example, /nobackupp10):

  ```
  pfe1% lfs quota -u username /nobackupp10
  ```

  or

  ```
  pfe1% lfs quota -u username /nobackup/username
  ```

  To display usage on each OST, add the -v option:

  ```
  pfe1% lfs quota -v -u username /nobackup/username
  ```

Pleiades Lustre Filesystems

Pleiades has several Lustre filesystems (/nobackupp[10-60]) that provide a total of about 3 PB of storage and serve thousands of cores. These filesystems are managed under Lustre software version 1.8.2.

Lustre filesystem configurations are summarized at the end of this article.

Which /nobackup should I use?

Once you are granted an account on Pleiades, you will be assigned to use one of the Lustre filesystems. You can find out which Lustre filesystem you have been assigned to by doing the following:

```
pfe1% ls -l /nobackup/your_username
lrwxrwxrwx 1 root root 19 Feb 23  2010 /nobackup/username -> /nobackupp30/username
```

In the above example, the user is assigned to /nobackupp30 and a symlink is created to point the user's default /nobackup to /nobackupp30.

TIP: Each Pleiades Lustre filesystem is shared among many users. To get good I/O performance for your applications and avoid impeding I/O operations of other users, read the articles: Lustre Basics and Lustre Best Practices.

Default Quota and Policy on /nobackup

Disk space and inodes quotas are enforced on the /nobackup filesystems. The default soft and hard limits for inodes are 75,000 and 100,000, respectively. Those for the disk space are 200GB and 400GB, respectively. To check your disk space and inodes usage and quota on your /nobackup, use the *lfs* command and type the following:

```
%lfs quota -u username /nobackup/username
Disk quotas for user username (uid xxxx):
   Filesystem  kbytes       quota   limit   grace   files   quota   limit   grace
/nobackup/username 1234  210000000 420000000    -     567   75000  100000       -
```

The NAS quota policy states that if you exceed the soft quota, an email will be sent to inform you of your current usage and how much of your grace period remains. It is expected that users will occasionally exceed their soft limit, as needed; however after 14 days, users who are still over their soft limit will have their batch queue access to Pleiades disabled.

If you anticipate having a long-term need for higher quota limits, please send a justification via email to support@nas.nasa.gov. This will be reviewed by the HECC Deputy Project Manager for approval.

For more information, see also, Quota Policy on Disk Space and Files.

NOTE: If you reach the hard limit while your job is running, the job will die prematurely without providing useful messages in the PBS output/error files. A Lustre error with code -122 in the system log file indicates that you are over your quota.

In addition, when a Lustre filesystem is full, jobs writing to it will hang. A Lustre error with code -28 in the system log file indicates that the filesystem is full. The NAS Control Room staff normally will send out emails to the top users of a filesystem asking them to clean up their files.

Important: Backup Policy

As the names suggest, these filesystems are not backed up, so any files that are removed *cannot* be restored. Essential data should be stored on Lou1-3 or onto other more permanent storage.

Configurations

In the table below, /nobackupp[10-60] have been abbreviated as p[10-60].

Pleiades Lustre Configurations

Filesystem	p10	p20	p30	p40	p50	p60
# of MDSes	1	1	1	1	1	1
# of MDTs	1	1	1	1	1	1
size of MDTs	1.1T	1.0T	1.2T	0.6T	0.6T	0.6T
# of usable inodes on MDTs	~235x10^6	~115x10^6	~110x10^6	~57x10^6	~113x10^6	~123x10^6
# of OSSes	8	8	8	8	8	8
# of OSTs	120	60	120	60	60	60
size/OST	7.2T	7.2T	3.5T	3.5T	7.2T	7.2T
Total Space	862T	431T	422T	213T	431T	431T
Default Stripe Size	4M	4M	4M	4M	4M	4M
Default Stripe Count	1	1	1	1	1	1

NOTE: The default stripe count and stripe size were changed on January 13, 2011. For directories created prior to this change, if you did not explictly set the stripe count and/or stripe size, the default values (stripe count 4 and stripe size 1MB) were used. This means that files created prior to January 13, 2011 had those old default values. After this date, directories without an explicit setting of stripe count and/or stripe size adopted the new stripe count of 1 and stripe size of 4MB. However, the old files in that directory will retain their old default values. New files that you create in these directories will adopt the new default values.

Lustre Best Practices

Lustre filesystems are shared among many users and many application processes, which causes contention for various Lustre resources. This article explains how Lustre I/O works, and provides best practices fro improving application performance.

How does Lustre I/O work?

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and object storage targets (OSTs) to perform the I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

Jobs being run on Pleiades content for shared resources in NAS's Lustre filesystem. The Lustre server can only handle about 15,000 remote procedure calls (RPCs, inter-process communications that allow the client to cause a procedure to be executed on the server) per second. Contention slows the performance of your applications and weakens the overall health of the Lustre filesystem. To reduce contention and improve performance, please apply the examples below to your compute jobs, while working in our high-end computing environment.

Best Practices

- **Avoid using *ls -l***

  The *ls -l* command displays information such as ownership, permission and size of all files and directories. The information on ownership and permission metadata is stored on the MDTs. However, the file size metadata is only available from the OSTs. So, the *ls -l* command issues RPCs to the MDS/MDT and OSSes/OSTs for every file/directory to be listed. RPC requests to the OSSes/OSTs are very costly and can take a long time to complete for many files and directories.

  - Use *ls* by itself if you just want to see if a file exists.

  - Use *ls -l filename* if you want the long listing of a specific file.

- **Avoid having a large number of files in a single directory**

Opening a file keeps a lock on the parent directory. When many files in the same directory are to be opened, it creates contention. It is better to split a huge number of files (in the thousands or more) into multiple sub-directories to minimize contention.

- **Avoid accessing small files on Lustre filesystems**

  Accessing small files on the Lustre filesystem is not efficient. If possible, keep them on an NFS-mounted filesystem (such as your home filesystem) or copy them from Lustre to /tmp on each node at the beginning of the job and access them from there.
- **Use a stripe count of 1 for directories with many small files**

  If you have to keep small files on Lustre, be aware that *stat* operations are more efficient if each small file resides in one OST. Create a directory to keep small files, set the stripe count to 1 so that only one OST will be needed for each file. This is useful when you extract source and header files (which are usually very small files) from a tarfile.

  ```
  pfe1% mkdir dir_name
  pfe1% lfs setstripe -s 1m -c 1 dir_name
  pfe1% cd dir_name
  pfe1% tar -xf tarfile
  ```

  If there are large files in the same directory tree, it may be better to allow them to stripe across more than one OST. You can create a new directory with a larger stripe count and copy the larger file to that directory. Note that moving files into that directory with the *mv* command will not change the strip count of the files. Files must be created in or copied to a directory to inherit the stripe count properties of a directory.

  ```
  pfe1% mkdir dir_count_4
  pfe1% lfs setstripe -s 1m -c 4 dir_count_4
  pfe1% cp file_count_1 dir_count_4
  ```

  If you have a directory with many small files (less than 100MB) and a few very large files (greater than 1GB), then it may be better to create a new subdirectory with a larger stripe count. Store just the large files and create symbolic links to the large files using the *symlink* command.

  ```
  pfe1%  mkdir bigstripe
  pfe1%  lfs setstripe -c 16 -s 4m bigstripe
  pfe1%  ln -s bigstripe/large_file  large_file
  ```
- **Use mtar for creating or extracting a tar file**

  A modified gnu tar command, */usr/local/bin/mtar*, is Lustre stripe aware and will create tar files or extract files with appropriately sized stripe counts. Currently, the number of streps is set to the number of gigabytes of the file.

- **Keep copies of your source on the Pleiades home filesystem and/or Lou**

  Be aware that files under /nobackup[p1,p2,p10-p60] are not backed up. Make sure that you have copies of your source codes, makefiles, and any other important files saved on your Pleiades home filesystem or on Lou, the NAS storage system.

- **Avoid accessing executables on Lustre filesystems**

  There have been a few incidents on Pleiades where users' jobs encountered problems while accessing their executables on /nobackup. The main issue is that the Lustre clients can become unmounted temporarily when there is a very high load on the Lustre filesystem. This can cause a bus error when a job tries to bring the next set of instructions from the inaccessible executable into memory.

  Executables run slower when run from the Lustre filesystem. It is best to run executables from your home filesystem on Pleiades. On rare occasions, running executables from the Lustre filesystem can cause executables to be corrupted. Avoid copying new executable over existing executables of the same within the Lustre filesystem. The copy causes a window of time (about 20 minutes) where the executable will not function. Instead, the executable should be accessed from your home filesystem during runtime.
- **Increase the stripe_count for parallel writes to the same file**

  When multiple processes are writing blocks of data to the same file in parallel, I/O performance is better for large files when the stripe_count is set to a larger value. The stripe count sets the number of OSTs the file will be written to. By default, the stripe count is set to 1. While this default setting provides for efficient access of metadata for example to support "ls -l"&emdash;large files should use stripe counts of greater than 1. This will increase the aggregate I/O bandwidth by using multiple OSTs in parallel instead of just one. A rule of thumb is to use a stripe count approximately equal to the number of gigabytes in the file.

  It is also better to make the stripe count be an integral factor of the number of processes performing the write in parallel so that one achieves load balance among the OSTs. For example, set the stripe count to 16 instead of 15 when you have 64 processes performing the writes.

- **Limit the number of processes performing parallel I/O**

  Given that the numbers of OSSes and OSTs on Pleiades are about a hundred or fewer, there will be contention if a huge number of processes of an application are involved in parallel I/O. Instead of allowing all processes to do the I/O, choose just a few processes to do the work. For writes, these few processes should collect the data from other processes before the writes. For reads, these few processes should read the data and then broadcast the data to others.

- **Stripe align I/O requests to minimize contention**

  Stripe aligning means that the processes access files at offsets that correspond to stripe boundaries. This helps to minimize the number of OSTs a process must communicate for each I/O request. It also helps to decrease the probability that multiple processes accessing the same file communicate with the same OST at the same time.

  One way to stripe-align a file is to make the stripe size the same as the amount of data in the write operations of the program.

- **Avoid repetitive *stat* operations**

  Some users have implemented logic in their scripts to test for the existence of certain files. Such tests generate *stat* requests to the Lustre server. When the testing becomes excessive, it creates a significant load on the filesystem. A workaround is to slow down the testing by adding *sleep* in the logic. For example, the following user script tests the existence of the files WAIT and STOP to decide what to do next.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
 end
```

  When neither the WAIT nor STOP file exists, the loop ends up testing for their existence as fast as possible (on the order of 5000 times per second). Adding a *sleep* inside the loop slows down the testing.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
  sleep 15
```

```
   end
```

- **Avoid multiple processes opening the same file(s) at the same time**

  On Lustre filesystems, if multiple processes try to open the same file(s), some processes will not able to find the file(s) and the job will fail.

  The source code can be modified to call the sleep function between I/O operations. This will reduce the occcurence of multiple access attempts to the same file from different processes simultaneously.

```
   100  open(unit,file='filename',IOSTAT=ierr)
        if (ierr.ne.0) then
         ...
        call sleep(1)
        go to 100
        endif
```

  When opening a read-only file in Fortran, use ACTION='read' instead of the default ACTION='readwrite'. The former will reduce contention by not locking the file.

```
open(unit,file='filename',ACTION='READ',IOSTAT=ierr)
```

- **Avoid repetitive open/close operations**

  Opening files and closing files incur overhead and repetitive open/close should be avoided.

  If you intend to open the files for read only, make sure to use **ACTION='READ'** in the open statement. If possible, read the files once each and save the results, instead of reading the files repeatedly.

  If you intend to write to a file many times during a run, open the file once at the beginning of the run. When all writes are done, close the file at the end of the run.

Reporting Problems

If you report performance problems with a Lustre filesystem, please be sure to include the time, hostname, PBS job number, name of the filesystem, and the path of the directory or file that you are trying to access.Your report will help us correlated issues with recorded performance data to determine the cause of efficiency problems.

Lustre Filesystem Statistics in PBS Output File

For a PBS job that reads or writes to a Lustre file system, a Lustre filesystem statistics block will appear in the PBS output file, just above the job's PBS Summary block. Information provided in the statistics can be helpful in determining the I/O pattern of the job and assist in identifying possible improvements to your jobs.

The statistics block lists the job's number of Lustre operations and the volume of Lustre I/O used for each file system. The I/O volume is listed in total, and is broken out by I/O operation size.

The following Metadata Operations statistics are listed:

- open/close of files on the Lustre file system
- stat/statfs are query operations invoked by commands such as "ls -l"
- read/write is the total volume of I/O in gigabytes

The following is an example of this listing:

```
=====================================================================
LUSTRE Filesystem Statistics
---------------------------------------------------------------------
  nbp10 Metadata Operations
       open     close      stat    statfs    read(GB)   write(GB)
       1057      1058      1394         0           2          14
Read   4KB   8KB  16KB  32KB  64KB  128KB  256KB   512KB   1024KB
         9     3     1     0     1      0      3       2      319
Write  4KB   8KB  16KB  32KB  64KB  128KB  256KB   512KB   1024KB
       138    13     1    11    36      9     21      37    12479
_____________________________________________________________________
Job Resource Usage Summary for 11111.pbspl1.nas.nasa.gov

    CPU Time Used           : 00:03:56
    Real Memory Used        : 2464kb
    Walltime Used           : 00:04:26
    Exit Status             : 0
```

The read and write operations are further broken down into buckets based on I/O block size. In the example above, the first bucket reveals that nine data reads occurred in blocks between 0 and 4 KB in size, three data reads ocurred with block sizes between 4 KB and 8 KB, and so on. The I/O block size data may be affected by library and system operations and, therefore, could differ from expected values. That is, small reads or writes by the program might be aggregated into larger operations, and large reads or writes might be broken into smaller pieces. If there are high counts in the smaller buckets, you should investigate the I/O pattern of the program for efficiency improvements.

Tips for Improving Lustre I/O

See Lustre Best Practices for multiple tips to improve the Lustre I/O performance of your jobs.